REMARKS

This communication responds to the Office Action mailed on November 3, 2004. Claim 10 is amended, no claims are canceled, and no claims are added. As a result, claims 1-28 are now pending in this Application. The claims are reproduced in Appendix A for convenient reference by the Examiner.

1. REAL PARTY IN INTEREST

The real party in interest of the above-captioned Application is the Assignee, Intel Corporation.

2. RELATED APPEALS AND INTERFERENCES

There are no interferences or appeals known to Appellants, Appellants' legal representative, or the Assignee that will directly affect or be directly affected by or have a bearing on the Board's decision in an appeal in this matter.

3. STATUS OF THE CLAIMS

Claims 1-28 are currently pending in the Application. Claims 24-28 have been allowed. Objections have been raised with respect to claims 3-9, 11-13, 15, 19, 20, 22, and 23 as being dependent on rejected base claims. Claims 1-2, 10, 14, 16-18, and 21 stand rejected, and the rejection is appealed herein.

4. STATUS OF AMENDMENTS

No amendments have been made subsequent to the amendment to claim 28 for reasons unrelated to patentability in conjunction with the Office Action Response filed on February 23, 2004. However, the Board is respectfully requested to consider the following amendment which provides clarity and consistency, and is not related to patentability:

10. (Currently Amended) A method of preparing a circuit model for simulation, the circuit model having a model size, ~~and the method~~ comprising:

merging a plurality of extended latch boundary components into a plurality of partitions having a partition size; and

maintaining a load balance within the plurality of partitions.

5. SUMMARY OF CLAIMED SUBJECT MATTER

This summary is presented in compliance with the requirements of Title 37 C.F.R. § 41.37(c)(1)(v), mandating a "concise explanation of the subject matter defined in each of the independent claims involved in the appeal ...". Nothing contained in this summary is intended to change the specific language of the claims described, nor is the language of this summary to be construed so as to limit the scope of the claims in any way.

Some embodiments of the invention are related to a method of preparing a circuit model for simulation. The method may include decomposing the circuit model (having a number of latches) into a plurality of extended latch boundary components, and partitioning the plurality of extended latch boundary components. (Application, Claim 1; FIG. 2; pg. 2, lines 6-8 and pg. 3, line 20 – pg. 4, line 8). The method may also include merging a plurality of extended latch boundary components into a plurality of partitions having a partition size and maintaining a load balance within the plurality of partitions. (Application, Claim 10; FIG. 5; and pg. 6, line 27 – pg. 7, line 9). The method may also include grouping a plurality of extended latch boundary components into a plurality of partitions and reducing the communication time within the plurality of partitions by adjusting the grouping. (Application, Claim 14; FIG. 6; and pg. 7, lines 10-24).

Some embodiments of the invention are related to a method of forming an extended latch boundary component. The method may include selecting a path having a first node selected from a group consisting of latches and primary outputs and a second node selected from a group consisting of latches and primary inputs, wherein the path can include a latch between the first node and the second node. (Application, Claim 16; FIG. 1; and pg. 3, lines 5–19.)

Some embodiments of the invention are related to a latch boundary component including a path comprising a plurality of first nodes selected from a group consisting of latches and primary outputs and a plurality of second nodes selected from a group consisting of latches and primary inputs, where the path can include a plurality of latches between the plurality of first nodes and the plurality of second nodes. (Application, Claim 17; FIG. 1; and pg. 3, lines 5–19.)

Some embodiments of the invention are related to a method of sharing a repeated circuit structure in a circuit model. The method may include expanding the repeated circuit structure once to form an expanded circuit structure and grafting the expanded circuit structure to the circuit model as needed. (Application, Claim 18; FIG. 7; and pg. 7, line 25 – pg. 8, line 5).

Some embodiments of the invention are related to a method of simulating a circuit model. The method may include partitioning a plurality of extended latch boundary components to form a plurality of partitions having a size, preparing a plurality of simulations from the plurality of partitions, and executing the plurality of simulations on a processing unit. (Application, Claim 21; FIG. 9; and pg. 8, line 14 – pg. 9, line 2).

6. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

6.1 Claim 18 stands rejected under 35 USC § 102(b) as being anticipated by Beausang et al. (U.S. 5,828,579; hereinafter "Beausang-3"). [Note: the references are numbered to coincide with the designations assigned by the Examiner in the Office Action.]

6.2 Claims 10, 16 and 17 stand rejected under 35 USC § 102(b) as being anticipated by Beausang et al. (U.S. 5,903,466; hereinafter "Beausang-1").

6.3 Claim 14 stands rejected under 35 USC § 103(a) as being unpatentable over Beausang-1 in view of Beausang-3.

6.4 Claims 1-2 and 21 stand rejected under 35 USC § 103(a) as being unpatentable over Beausang-1 in view of Beausang et al. (U.S. 5,949,692; hereinafter "Beausang-2").

7. ARGUMENT

7.1 The Applicable Law

Anticipation under 35 USC § 102 requires the disclosure in a single prior art reference of each element of the claim under consideration. *See Verdegaal Bros. V. Union Oil Co. of California,* 814 F.2d 628, 631, 2 USPQ 2d 1051, 1053 (Fed. Cir. 1987). It is not enough, however, that the prior art reference discloses all the claimed elements in isolation. Rather, "[a]nticipation requires the presence in a single prior reference disclosure of each and every element of the claimed invention, *arranged as in the claim*." *Lindemann Maschinenfabrik GmbH v. American Hoist & Derrick Co.*, 730 F.2d 1452, 221 USPQ 481, 485 (Fed. Cir. 1984) (citing *Connell v. Sears, Roebuck & Co.*, 722 F.2d 1542, 220 USPQ 193 (Fed. Cir. 1983)) (emphasis added). "The *identical invention* must be shown in as complete detail as is contained in the ...

claim." *Richardson v. Suzuki Motor Co.*, 868 F.2d 1226, 1236, 9 USPQ2d 1913, 1920 (Fed. Cir. 1989); MPEP § 2131 (emphasis added).

The Examiner has the burden under 35 U.S.C. § 103 to establish a *prima facie* case of obviousness. *In re Fine*, 837 F.2d 1071, 1074, 5 U.S.P.Q.2d (BNA) 1596, 1598 (Fed. Cir. 1988). The M.P.E.P. contains explicit direction to the Examiner that agrees with the *In re Fine* court:

> In order for the Examiner to establish a *prima facie* case of obviousness, three base criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations. The teaching or suggestion to make the claimed combination and the reasonable expectation of success must both be found in the prior art, and not based on applicant's disclosure. *M.P.E.P.* § 2142 (citing *In re Vaeck* , 947 F.2d 488, 20 U.S.P.Q.2d (BNA) 1438 (Fed. Cir. 1991)).

The requirement of a suggestion or motivation to combine references in a *prima facie* case of obviousness is emphasized in the Federal Circuit opinion, *In re Sang Su Lee*, 277 F.3d 1338; 61 U.S.P.Q.2D 1430 (Fed. Cir. 2002), which indicates that the motivation must be supported by evidence in the record.

The test for obviousness under § 103 must take into consideration the invention as a whole; that is, one must consider the particular problem solved by the combination of elements that define the invention. *Interconnect Planning Corp. v. Feil*, 774 F.2d 1132, 1143, 227 U.S.P.Q. 543, 551 (Fed. Cir. 1985). References must be considered in their entirety, including parts that teach away from the claims. See MPEP § 2141.02.

7.2 The References

Beausang-1: discloses a scan insertion process for complex circuit synthesis that has a reduced set of constraint driven compiler optimizations. See Beausang-1, Col. 4, lines 46-48. A three-tiered performance optimization is used within the insertion process: the first tier optimizes design for test elements; the second tier optimizes all elements in the design (in addition to implementing the first tier); and the third tier performs sequential optimization, as well as local optimization (in addition to implementing the second tier). See Beausang-1, Col. 4, lines 50–59.

Beausang-2: describes a synthesis process of scan insertion/replacement and routing. See Beausang-2, Col. 31, lines 48-56. Scan resources may be inserted into an integrated circuit design organized into hierarchical modules. See Beausang-2, Col. 3, lines 52-54.

Beausang-3: is directed toward a scan chain design database that defines a hierarchical circuit design as a netlist of logic cells, some of which contain scan structure. See Beausang-3, Col. 6, lines 47-66. Inferred scan structures may be recognized and replaced with explicit scan structures. See Beausang-3, Col. 7, lines 7-28.

7.3 Discussion of the Rejections

7.3.1 The Rejections Under § 102:

Claim 18 was rejected under 35 USC § 102(b) as being anticipated by Beausang-3. Claims 10, 16 and 17 were rejected under 35 USC § 102(b) as being anticipated by Beausang-1. The Appellants do not admit that Beausang-1 or Beausang-3 are prior art, and reserve the right to swear behind these references at a later date. In addition, because the Appellants assert that the Office has not shown that Beausang-1 or Beausang-3 discloses the identical invention as claimed, the Appellants respectfully traverse these rejections of the claims.

With respect to claim 18, it is noted that Beausang-3 is directed toward a scan chain design database that defines a hierarchical circuit design as a netlist of logic cells, some of which contain scan structure. See Beausang-3, Col. 6, lines 47-66. While the assertion is made in the Office Action that Beausang-3 Figs. 6A, 6B, and 7B illustrate various elements of the claim, it is respectfully noted that the cited elements are actually shown as part of creating balanced scan chains, and not sharing a repeated circuit structure in a circuit model, as claimed by the Appellants.

FIGs. 6A and 6B of Beausang-3 show before and after snapshots of synthesis results for two different chains having a single clock domain, while FIGs. 7A and 7B show before and after snapshots of synthesis results for two different chains having a mixed clock edge design. See Beausang-3, Col. 33, lines 5-50. The Appellants were unable to locate the activities of "expanding the repeated circuit structure" (there is no repeated structure shown) and "grafting

the expanded structure" (since there is no expanded, repeating structure) within the bounds of Beausang-3, as claimed by the Appellants.

The Office attempts to explain similarities between Beausang-3 and the claimed embodiment by asserting that the insertion of Beausang-3's latch 713 into level 712 (see Beasang-3, FIGs. 10A-10B) represents "expansion of a repeated circuit structure." A similar assertion is made with respect to the activity represented by block 335 of Beausang-3's FIG. 2A (where an unused chain is added to the scan plan – the unused chain being user-defined and having no scan structure associated with it). See Beausang-3, Col. 27, lines 15-23. However, neither of these assertions support the limitation of "expanding the repeated circuit structure once to form an expanded circuit structure" claimed by the Appellants, since no method of sharing a repeated circuit structure is shown.

As to claim 10, it appears that an element claimed by the Appellants (i.e., "merging a plurality of extended latch boundary components into a plurality of partitions") was not addressed in the Office Action, and the Appellants were unable to find any indication of their existence in Beausang-1. The Office attempts to explain that the mere presence of equal numbers of latch elements 307 and 321 in Beausang-1, FIG. 3B equates to "maintaining a load balance within the plurality of partitions", as claimed by the Appellants. However, this is not the case. As noted in Beausang-1, "a scan replacement process ... replaces the non-scan memory cells 307a-e of unit 301 with scannable memory cells 320a-e (FIG. 3B) ...". Thus, there is no load balance maintenance whatsoever; the process described by Beausang-1 is one of replacing one set of elements with another, and not balancing between partitions. See Beausang-1, Col. 9, lines 40-43.

As to claim 16, it is respectfully noted that there is no indication that the circuit combination shown in FIG. 13B of Beausang-1 is a result of the claimed process (i.e., a method of forming an extended latch boundary component comprising "selecting a path having a first node selected from a group consisting of latches and primary outputs and a second node selected from a group consisting of latches and primary inputs").

As to claim 17, there do not appear to be either "primary inputs" (e.g., only a single primary input 307 is shown in FIG. 3A) or "primary outputs" (none are shown) as set forth in the

claim (i.e., a path comprising a plurality of first nodes selected from a group consisting of latches and primary outputs and a plurality of second nodes selected from a group consisting of latches and primary inputs, where the path can include a plurality of latches between the plurality of first nodes and the plurality of second nodes").

A question regarding what may comprise a "primary input" or "primary output" was raised in the Office Action. These terms are well-understood by those of skill in the art of circuit synthesis. Numerous references make use of these terms. For example, as explained in *Wireplanning in Logic Synthesis*:

> "A logic circuit L is a 3-tuple (I, O, F). I is a set of primary input pins, or simply primary inputs. O is a set of primary output pins, or simply primary outputs. Each element of I an O is a binary variable." *Wireplanning in Logic Synthesis*, ICCAD98, pg. 26 (attached hereto as part of Appendix B).

Similarly, as noted in *Logic Synthesis Preserving High-Level Specification*:

> "Let S be a single output combinational circuit of multi-valued blocks specified by a directed acyclic graph H. The sources and the sink of H correspond to primary inputs and the output of S. Each non-source node of H corresponds to a multivalued block computing a multi-valued function of multivalued arguments. Each node of n of H is associated with a multi-valued variable A. If n is a source of H, then the corresponding variable specifies values taken by the corresponding primary input of S. If n is a non-source node of S then the corresponding variable describes the values taken by the output of the block specified by n. If n is a source (respectively the sink), then the corresponding variable is called a primary input variable (respectively primary output variable)." *Logic Synthesis Preserving High-Level Specification*, E.Goldberg (Cadence Berkeley Labs, USA), located at http://eigold.tripod.com/papers/iwls-2004.pdf (attached hereto as part of Appendix B).

Finally, while it is asserted in the Office Action that Beausang-1 teaches the combination of "a plurality of first nodes (Figure 15A ...), with a plurality of output latches (Figure 3B)," the Appellants can find no logical connection between these two elements within the bounds of Beausang-1. The cited portions of this reference (FIGS. 13B, 15A, and 3B) are directed to a scan insertion process to (i) move a load from a Q logic output to a $\overline{Q}$ logic output to adjust the load input phase; (ii) move a load to a logically-equivalent driving input so the original driver can be downsized; and (iii) replace HDL-specified, non-scan memory cells with DFT scannable

memory cells. Beausang-1, Col. 6 line 15 – Col. 7, line 6; and Col. 24, line 58 – Col. 25, line 24. Thus, it does not appear that it is possible for the cited combination to exist according to the teachings of Beausang-1.

"The *identical invention* must be shown in as complete detail as is contained in the ... claim." *Richardson v. Suzuki Motor Co.*, 868 F.2d 1226, 1236, 9 USPQ2d 1913, 1920 (Fed. Cir. 1989); MPEP § 2131 (emphasis added). Therefore, since what is disclosed by Beausang-1 and Beausang-3 is not identical to the subject matter of the embodiments claimed, the rejection of claims 10 and 16-18 under § 102(b) is improper. Reconsideration and allowance are respectfully requested.

7.3.2 The Rejections Under § 103:

Claims 1, 2 and 21 were rejected under 35 USC § 103(a) as being unpatentable over Beausang-1 in view of Beausang-2. Claim 14 was rejected under 35 USC § 103(a) as being unpatentable over Beausang-1 in view of Beausang-3. First, the Appellants do not admit that Beausang-1, Beausang-2, or Beausang-3 are prior art, and reserve the right to swear behind these references in the future. Second, since a *prima facia* case of obviousness has not been established in each case, the Appellants respectfully traverse these rejections.

No proper *prima facie* case of obviousness has been established because (1) combining the references does not teach all of the limitations set forth in the claims, (2) there is no motivation to combine the references, and (3) combining the references provides no reasonable expectation of success. Each of these points will be explained in detail, as follows.

Combining References Does Not Teach All Limitations: First, with respect to independent claims 1 and 21, no combination suggested in the Office Action will render all of the claim limitations. It is admitted by the Office that Beausang-1 does not disclose "partitioning the plurality of extended latch boundary components" as claimed by the Appellants. Neither does Beausang-2.

Beausang-2 is directed to a synthesis process of scan insertion/replacement and routing. See Beausang-2, Col. 31, lines 48-56. The Appellants were unable to find any reference within the bounds of Beausang-2 to partitioning "extended latch boundary components." Rather, the

figures cited in the Office Action (Beausang-2, FIGs. 2A and 6A-14B) illustrate various portions of scan chains; partitioning extended latch boundary components to serve any of the purposes noted by the Appellants in the Application is not taught. The text cited in the Office Action (Beausang-2, Col. 26, line 50 – Col. 27, line 17) refers to selecting "partition blocks" to balance scan chains, and not to parititioning extended latch boundary components. The logic cells of the database remain unaltered. Thus, independent claims 1 and 21 are nonobvious. This conclusion applies with even greater force respecting dependent claim 2, since any claim depending from a nonobvious independent claim is also nonobvious. See M.P.E.P. § 2143.03.

Second, with respect to independent claim 14, the Office admits that Beausang-1 does not disclose "grouping a plurality of extended latch boundary components into a plurality of partions", as claimed by the Appellants. Neither does Beausang-3.

Beausang-3 is also directed to a synthesis process of scan insertion/replacement and routing. See Beausang-3, Col. 32, lines 22-42. The Appellants were unable to find any reference within the bounds of Beausang-3 to "grouping a plurality of extended latch boundary components into a plurality of partions." Rather, the cited segments of Beausang-3 (Beausang-3, Col. 12, lines 45-55) serve to define scan groups, scan links, and scan chains; grouping extended latch boundary components to serve any of the purposes noted by the Appellants in the Application is not taught. Thus, independent claim 14 is nonobvious.

No Motivation to Combine References: The Office asserts that one would be motivated to combine Beausang-1 with Beausang-2 or Beausang-3 so that a designer would be able to "sign off" his or her work at the completion of module design, without later disruption. See Office Action, Paper 10-24-04, Pg. 8. However, as pointed out in the Office Action, Beausang-1 already provides an enabling solution.

The assertion is made by the Office that, nevertheless, one of skill in the art "would have been motivated to find a better solution to decrease the amount of time required to compile a design." It is further asserted that it would therefore have been obvious to combine Beasang-1 with either Beausang-2 or Beausang-3 so that "the IC designer can now 'sign off' his or her work at the completion of the module design" so that completed/optimized modules do not later have to be disrupted. However, the Appellants do not understand that enabling designer sign off

necessarily satisfies the motivation noted in the Office Action. In fact, the combination may require *extra* time to finalize individual module designs, potentially creating a *longer* compilation time, since each module must now be perfected by its individual designer prior to top level analysis. See Beausang-2, Col. 3, lines 3-11 and Beausang-3, Col. 3, lines 4-12. Thus, there is no motivation to combine Beausang-1 with either Beausang-2 or Beausang-3.

Finally, it is noted in the Office Action that all three references have the same inventor, James Beausang, and that this somehow constitutes a motivation to combine them. However, combining references in a way that prevents achieving the goals expressed in the references does not constitute a motivation to combine them.

Since Beausang-1 teaches away from the suggested combinations, the use of unsupported assertions in the Office Action does not satisfy the explicit requirements needed to demonstrate motivation as set forth by the *In re Sang Su Lee* court. Therefore, the Examiner appears to be using personal knowledge, and is again respectfully requested to submit an affidavit as required by 37 C.F.R. § 1.104(d)(2).

No Reasonable Expectation of Success: As has been previously noted, modifying Beausang-1 to implement independent completion of modules by various designers may create additional barriers to overall design completion, without providing a "system that can reduce the time required to perform circuit synthesis ..." See Beausang-1, Col. 4, lines 18-19. Introducing a human element into an electronic design process rarely speeds anything up; in fact, "[d]esign, checking and testing of large scale integrated circuits are so complex that the use of programmed computer systems are required for realization of normal circuits." See Beausang-1, Col. 1, lines 21-23.

In addition, as noted above, several elements of claims 1-2, 14, and 21 are not provided by any of the cited references. Thus, there is no reasonable expectation that any combination of Beausang-1, Beausang-2, and Beausang-3 will be unable to provide the missing elements, such as "decomposing", "partitioning", and "grouping" extended latch boundary components, as claimed by the Appellants.

The test for obviousness under § 103 must take into consideration the invention as a whole; that is, one must consider the particular problem solved by the combination of elements

that define the invention. *Interconnect Planning Corp. v. Feil*, 774 F.2d 1132, 1143, 227 U.S.P.Q. 543, 551 (Fed. Cir. 1985). References must be considered in their entirety, including parts that teach away from the claims. See MPEP § 2141.02. The fact that references can be combined or modified does not render the resultant combination obvious unless the prior art also suggests the desirability of the combination. *In re Mills*, 16 USPQ2d 1430 (Fed. Cir. 1990); M.P.E.P. § 2143.01.

Therefore, since there is no evidence in the record to support disclosure by either Beausang-1, Beausang-2, or Beausang-3 of "decomposing", "partitioning", and "grouping" extended latch boundary components, since there is no motivation to supply the missing elements (since the references teach away from such a combination), and since no reasonable expectation of success arises, a *prima facie* case of obviousness has not been established with respect to independent claims 1, 14, and 21. This conclusion also applies to dependent claim 2, since any claim depending from a nonobvious independent claim is also nonobvious. It is therefore respectfully requested that the rejections of claims 1-2, 14, and 21 under 35 U.S.C. § 103 be reconsidered and withdrawn.

8. SUMMARY

It is respectfully submitted that no *prima facie* case of anticipation under 35 U.S.C. §102, nor of obviousness under 35 U.S.C. §103 has been established by the Office. Therefore, it is respectfully requested that the rejections of claims 1-2, 10, 14, 16-18, and 21 be reconsidered and withdrawn. The Appellants respectfully submit that all of the claims are in condition for allowance and notification to that effect is earnestly requested. The Examiner is invited to telephone the Appellants' attorney, Mark Muller at (210) 308-5677, or the undersigned attorney at (612) 373-6970, to facilitate prosecution of this Application. If necessary, please charge any additional fees or credit overpayment to Deposit Account No. 19-0743.

Respectfully submitted,

MANPREET S. KHAIRA ET AL.

By their Representatives,

SCHWEGMAN, LUNDBERG, WOESSNER & KLUTH, P.A.
Attorneys for Intel Corporation
P.O. Box 2938
Minneapolis, Minnesota 55402
(612) 373-6970

Date January 3, 2005

By _____
Charles E. Steffey
Reg. No. 25,179

CERTIFICATE UNDER 37 CFR 1.8: The undersigned hereby certifies that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail, in an envelope addressed to: Mail Stop AF, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on this 3rd day of January 2005.

Dennis J. Kamph
Name

_____
Signature

APPENDIX A - CLAIMS

1. (Original) A method of preparing a circuit model for simulation comprising:

decomposing the circuit model having a number of latches into a plurality of extended latch boundary components; and

partitioning the plurality of extended latch boundary components.

2. (Original) The method of claim 1, wherein decomposing a circuit model having a number of latches into a plurality of extended latch boundary components comprises:

decomposing at least one of a plurality of hierarchical cells into one of the plurality of extended latch boundary components.

3. (Original) The method of claim 2, wherein partitioning the plurality of extended latch boundary components comprises:

using a constructive bin-packing heuristic to partition the plurality of extended latch boundary components.

4. (Original) The method of claim 3, wherein using a constructive bin-packing heuristic to partition the plurality of extended latch boundary components comprises:

constructing a plurality of seeds from the plurality of extended latch boundary components; and

merging the plurality of extended latch boundary components with the plurality of seeds.

5. (Original) The method of claim 1, wherein decomposing a circuit model having a number of latches into a plurality of extend latch boundary components comprises:

identifying an extended latch boundary component that meets a size constraint for at least one of a plurality of hierarchical cells.

6. (Original) The method of claim 5, wherein partitioning the plurality of extended latch boundary components comprises:

grouping the plurality of extended latch boundary components into a plurality of partitions by approximately equalizing the number of latches in each of the plurality of partitions.

7. (Original) The method of claim 1, wherein partitioning the plurality of extended latch boundary components comprises:

grouping the plurality of extended latch boundary components to form a plurality of partitions, each of the plurality of partitions having a size.

8. (Original) The method of claim 7, wherein partitioning the plurality of extended latch boundary components comprises:

partitioning the plurality of extended latch boundary components by approximately equalizing the number of latches in each of the plurality of partitions, approximately equalizing the latches that are activated in each of the plurality of partitions, and approximately equalizing the size of each of the plurality of partitions.

9. (Original) The method of claim 1, wherein partitioning the plurality of extended latch boundary components comprises:

attempting to partition the plurality of extended latch boundary components based on activity load balancing.

10. (Currently Amended) A method of preparing a circuit model for simulation, the circuit model having a model size, ~~and the method~~ comprising:

merging a plurality of extended latch boundary components into a plurality of partitions having a partition size; and

maintaining a load balance within the plurality of partitions.

11. (Original) The method of claim 10, further comprising:

reducing circuit overlap within the plurality of partitions.

12. (Original) The method of claim 11, further comprising:

adjusting the load balance to obtain a partition size of less than about 110% of the model size.

13. (Original) The method of claim 12, further comprising:

adjusting the load balance to obtain a partition size of less than about 120% of the model size.

14. (Original) A method of preparing a circuit model for a simulation having a total simulation time, the method comprising:

grouping a plurality of extended latch boundary components into a plurality of partitions; and

reducing the communication time within the plurality of partitions by adjusting the grouping.

15. (Original) The method of claim 14, further comprising:

reducing the communication time within the plurality of partitions to less than about ten percent of the total simulation time by adjusting the grouping.

16. (Original) A method of forming an extended latch boundary component comprising:

selecting a path having a first node selected from a group consisting of latches and primary outputs and a second node selected from a group consisting of latches and primary inputs, wherein the path can include a latch between the first node and the second node.

17. (Original) A latch boundary component comprising:

a path comprising a plurality of first nodes selected from a group consisting of latches and primary outputs and a plurality of second nodes selected from a group consisting of latches and primary inputs, where the path can include a plurality of latches between the plurality of first nodes and the plurality of second nodes.

18. (Original) A method of sharing a repeated circuit structure in a circuit model, the method comprising:

expanding the repeated circuit structure once to form an expanded circuit structure; and

grafting the expanded circuit structure to the circuit model as needed.

19. (Original) The method of claim 18, wherein grafting the expanded circuit structure to the circuit model as needed comprises:

copying a table representing the expanded circuit structure into the circuit model.

20. (Original) The method of claim 18, wherein grafting the expanded circuit structure to the circuit model as needed comprises:

altering a table representing the circuit model to add the expanded circuit structure.

21. (Original) A method of simulating a circuit model, the method comprising:

partitioning a plurality of extended latch boundary components to form a plurality of partitions having a size;

preparing a plurality of simulations from the plurality of partitions; and

executing the plurality of simulations on a processing unit.

22. (Original) The method of claim 21, further comprising:

adjusting the size of the plurality of partitions.

23. (Original) The method of claim 22, wherein executing the plurality of simulations on a processing unit comprises:

executing the plurality of simulations on a plurality of distributed processors.

24. (Original) A computer system comprising:

a processor unit;

a dicing unit operably coupled to the processor unit, capable of executing on the processor unit, and capable of decomposing a circuit model into a plurality of extended latch boundary components, and capable of partitioning the plurality of extended latch boundary components; and

a simulation unit operably coupled to the dicing unit and the processor unit, and capable of executing on the processor unit.

25. (Original) The computer system of claim 24, wherein the processor unit is a plurality of distributed processor units.

26. (Original) The computer system of claim 25, wherein the dicing unit is capable of load balancing.

27. (Original) The computer system of claim 26, wherein the dicing unit is capable of activity load balancing.

28. (Previously Presented) A computer-readable medium having computer-executable instructions, wherein the computer-executable instructions, when accessed, result in a machine performing:

partitioning a circuit model into a plurality of cells arranged in a hierarchy; and

mapping a plurality of extended latch boundary components into the circuit model by finding each cell in the plurality of cells that is highest in the hierarchy such that a single extended latch boundary component satisfying a given size constraint can be mapped into the cell.

APPENDIX B - EVIDENCE

INSERT #1: WIREPLANNING IN LOGIC SYNTHESIS, 8 pgs.

INSERT #2: LOGIC SYNTHESIS PRESERVING HIGH-LEVEL SPECIFICATION, 8 pgs.

APPENDIX C – RELATED PROCEEDINGS

[No Related Proceedings are Known to the Appellants' Representative]

APPENDIX B

Logic synthesis preserving high-level specification

E.Goldberg (Cadence Berkeley Labs, USA),

Abstract. *In this paper we develop a method of logic synthesis that preserves high-level structure of the circuit to be synthesized. This method is based on the fact that two combinational circuits implementing the same "high-level" specification can be efficiently checked for equivalence. Hence, logic transformations preserving a predefined specification can be made efficiently. We introduce the notion of toggle equivalence of Boolean functions and show that toggle equivalence can be used for making gate level transformations that preserve a predefined specification. We describe a practical procedure for checking toggle equivalence of two Boolean circuits and give experimental data about its performance.*

1. Introduction

In a typical design flow, by the time a circuit is passed to a logic synthesis procedure, the "high-level" structure of this circuit is lost. Suppose, for example, that a combinational circuit is initially described as a network of multi-valued blocks. After encoding all the multi-valued variables, this network is replaced with a Boolean circuit that is optimized using a set of local transformations that ignore the original high-level structure of the circuit.

The flaw of the approach above is that in the case of a poor choice of encodings for multi-valued variables, a synthesis procedure using local transformations will not be able to "correct" these encodings. On the other hand, finding good encodings at the level of multi-valued blocks is hard and so the probability of generating bad encodings is high. A possible solution to the problem is to perform logic transformations that re-encode multi-valued variables "implicitly" at the gate level. We will call such transformations ***High-Level structure aware Logic Synthesis (HLLS)***. This is because re-encoding of multi-valued variables implicitly, essentially means synthesizing a circuit that is a different implementation of the same "specification" as the original circuit. An HLLS procedure can be used as an extra optimization step taken before using logic synthesis based on local transformations.

In this paper we introduce a method of HLLS for combinational circuits. To design an HLLS procedure one has to solve two problems. The first problem is to verify the correctness of logic transformations. A "regular" logic synthesis procedure makes local transformations so the equivalence checking of the original and optimized circuits is usually not an issue. On the other hand, an HLLS procedure performs "non-local" synthesis transformations, so verification of the correctness of such transformations may pose a problem. This problem was addressed in [3], [4] where it was shown that if two Boolean circuits have a common specification (CS), their equivalence checking is "easy". Informally, circuits N_1 and N_2 have a CS if they can be considered as two different implementations of a circuit S of multi-valued gates further referred to as blocks. (S is called a CS of N_1 and N_2). In [3][4] it was proven that given a CS S of circuits N_1 and N_2, there is an equivalence checking procedure whose complexity is linear in the number of blocks of S and exponential in the granularity of S (the "size" of the largest block of S). An example of circuits N_1,N_2 having a common specification of three blocks is shown in Fig. 1. The specification itself is shown on the left. Here $N_1(G_k)$ and $N_2(G_k)$ are subcircuits of N_1 and N_2 respectively implementing the same block of specification.

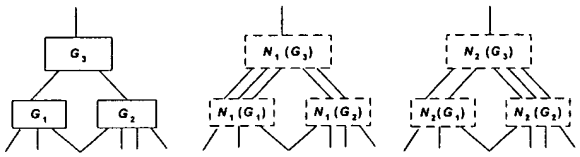

Figure 1 Circuits N_1 and N_2 with a common specification of three blocks

The second problem one has to solve in HLLS, is to find a way to preserve a predefined specification by making transformations at the gate level. Solving this problem is the focus of this paper. We introduce the notion of toggle equivalence of multi-output Boolean functions that is a generalization of regular functional equivalence. We show that circuits N_1 and N_2 have a CS if they can be partitioned into toggle equivalent subcircuits (see Section 4.). This result suggests a simple way to perform HLLS. Suppose that a specification S of circuit N_1 to be optimized is specified by partitioning the latter into subcircuits. Then, if we replace subcircuits of N_1 with "better" subcircuits that are toggle equivalent to replaced ones, we produce a circuit N_2 that implements the same specification S as N_1. To be viable, HLLS needs an efficient algorithm for checking toggle equivalence. We describe such an algorithm and demonstrate its efficiency on benchmark circuits .

The paper is structured as follows. In Section 2 we formally define the notion of a common specification. In

Section 3 we reformulate the algorithm of equivalence checking from [3] removing some redundancy. Section 4 introduces the notion of toggle equivalence of Boolean functions. The relation between the notions of toggle equivalence and common specification is shown in Section 5. In Section 6 we describe a method of HLLS. The relation of HLLS to other synthesis procedures is discussed in Section 7. The description of a procedure for checking toggle equivalence and its performance on benchmark circuits are given in Section 8. Finally, we draw some conclusions in Section 9.

2. Definition of common specification

In this section, we formally define the notion of a common specification of Boolean circuits. Let S be a single output combinational circuit of multi-valued blocks specified by a directed acyclic graph H. The sources and the sink of H correspond to primary inputs and the output of S. Each non-source node of H corresponds to a multi-valued block computing a multi-valued function of multi-valued arguments. Each node of n of H is associated with a ***multi-valued variable*** A. If n is a source of H, then the corresponding variable specifies values taken by the corresponding primary input of S. If n is a non-source node of S then the corresponding variable describes the values taken by the output of the block specified by n. If n is a source (respectively the sink), then the corresponding variable is called a ***primary input variable*** (respectively ***primary output variable)***. We will use the notation $C=G(A_1,A_2,\ldots, A_k)$ to indicate that a) the output of a block G is associated with a variable C; b) the function computed by the block G is $G(A_1,A_2,\ldots, A_k)$; c) only k nodes of H are connected to the node n in H and outputs of these nodes are associated with variables $A_1,A_2,\ldots, A_k$.

Denote by $D(A)$ the ***domain*** of the variable A associated with a node of H. The value of $|D(A)|$ is called the ***multiplicity*** of A. If the multiplicity of every variable A of S is equal to 2 then S is a ***Boolean circuit***.

Now we introduce the notion of a specification of a single output Boolean circuit N. Informally, a multi-valued circuit S is a specification of N if N can be obtained from S by picking proper encodings of internal variables of S. In the following exposition any multi-valued network is called a specification.

Definition 1. Let $D(A)=\{a_1,\ldots,a_t\}$ be the domain of a variable A of S. Denote by $q(A)$ a Boolean encoding of the values of $D(A)$ which is a mapping $q:D(A)\rightarrow\{0,1\}^m$ such that $a_i \neq a_j \Rightarrow q(a_i) \neq q(a_j)$. The value of $q(a_i)$, $a_i \in D(A)$ is called the ***code*** of a_i. Denote by $length(q(A))$ the number of bits in $q(a_i)$ i.e. the value of m. Denote by $v(A)$ the set of m ***coding Boolean variables***.

In this paper, we make the assumptions below about specifications and implementations

Assumption 1. A specification contains only one output. All primary input variables and the primary output variable of a specification are Boolean.

Assumption 2. Every gate of a Boolean circuit (implementation) has two inputs and one output. Every multi-valued block of a specification has only one output but the number of inputs of a block is not fixed.

Assumption 3. If A_i and A_j are two different variables of a specification, then $v(A_i) \cap v(A_j) = \varnothing$.

Definition 2. Let $C=G(A_1,A_2,..,A_n)$ be a block of specification S. Let $q(A_1),..,q(A_n),q(C)$ be encodings of variables $A_1,A_2,..,A_n$ and C respectively. A Boolean circuit N is said to ***implement the block G*** if N implements the completely specified Boolean function $q(C)=G(q(A_1),..,q(A_n))$ whose truth table is obtained from that of G by replacing values of $A_1,..,A_n$,C with their codes.

Definition 3. Let S be a multi-valued circuit. A single output Boolean circuit N is said to ***implement the specification S***, if N can be built from S by the following two rules.

1) Each block G of S is replaced with its implementation (denote it by $N(G)$).

2) Let the output of block G_1 (specified by variable C) be connected to an input of block G_2 (specified by the same variable C) in S. Then the outputs of the circuit $N(G_1)$ are properly connected to inputs of $N(G_2)$. Namely, if a primary output of $N(G_1)$ connected to an input of $N(G_2)$ these input and output are specified by the same coding variable of $v(C)$

Remark 1. It is important to emphasize that the fact that a circuit N has a specification S does not necessary mean that N *is produced* from S by encoding it multi-valued variables. It just means that N *can* be produced from S.

Remark 2. Let N be an implementation of a specification S. Let p be the largest number of gates used in an implementation of a multi-valued block of S in N. We will say that S is a specification of ***granularity*** p for N.

Definition 4. Let N_1, N_2 be two functionally equivalent single output Boolean circuits. Let N_1, N_2 implement a specification S. Then S is called ***a common specification*** (CS) of N_1 and N_2.

Definition 5. Let S be a CS of N_1,N_2. Let p_1 (respectively p_2) be the granularity of S with respect to N_1 (respectively N_2). Then we will say that S is a CS of N_1,N_2 of ***granularity*** $p = max(p_1,p_2)$.

3. Equivalence checking with a known common specification

In this section, we recall the equivalence checking algorithm of [3] and give a slightly modified version of it.

Let N_1 and N_2 be Boolean circuits with a CS S. Let G_k be a block of S. We will denote by $N_1(G_k)$ and $N_2(G_k)$ the implementations of the block G_k in N_1 and N_2 respectively.

Definition 6. Let S be a CS of N_1 and N_2. The ***topological level*** of a block G_k in a specification S is the length of the longest path from a primary input of S to G_k. (The length of a path is measured in the number of blocks on it. The topological level of a primary input is assumed to be 0.) Denote by ***level(G_k)*** the topological level of G_k in S. Denote by *level*($N_i(G_k)$) the topological level of implementation of block G_k in N_i, i=1,2 that is assumed to be equal to *level*(G_k).

Definition 7. Let $N_1(G_k)$ and $N_2(G_k)$ be implementations of a multi-valued block G_k whose output is associated with variable C. Let $q_1(C)$ and $q_2(C)$ be encodings of the variable C used in implementations $N_1(G_k)$ and $N_2(G_k)$. Function $Cf(v_1(C),v_2(C))$ is called a ***correlation function*** of encodings $q_1(C),q_2(C)$ if

a) $Cf(z_1, z_2)$=1 for any assignment z_1 to $v_1(C)$ and z_2 to $v_2(C)$ such that $z_1=q_1(c)$ and $z_2=q_2(c)$ where $c \in D(C)$.

b) Otherwise $Cf(z_1, z_2)$=0.

Definition 8. Let N be a Boolean circuit. Denote by $v(N)$ the set of Boolean variables associated with the output of gates of N. Denote by Sat(v(N)) the Boolean function such that $Sat(z)$=1 iff the assignment z to variables $v(N)$ is "possible" i.e consistent. For example, if circuit N consists of just one AND gate $y=x_1 \wedge x_2$, then $v(N)=\{y,x_1,x_2\}$ and $Sat(v(N))= (\sim x_1 \vee \sim x_2 \vee y) \wedge (x_1 \vee \sim y) \wedge (x_2 \vee \sim y)$.

Definition 9. Let f be a Boolean function. We will say that function f^* is obtained from f by existentially quantifying away variable x if $f^* = f(\ldots,x=0,\ldots) \vee f(\ldots,x=1,\ldots)$.

In [3] it was shown that if N_1 and N_2 have a CS S, one can check them for equivalence in the time linear in the number of blocks of S and exponential in the granularity of S. The essence of that algorithm is to compute so-called filtering and correlation functions in topological order. Here we give a modified version of this algorithm. The modification is that we discard computation of filtering functions from the algorithm.

Here is an informal proof that computation of filtering functions is not necessary. Let C be the variable associated with the output of a block G_k of S. From definitions of filtering (denoted by Ff) and correlation functions given in [3] it follows that

$Ff(v_1(C)) \wedge Ff(v_2(C)) \wedge Cf(v_1(C),v_2(C)) = Cf(v_1(C),v_2(C))$.

So filtering functions can be dropped from the proof of Proposition 7 of [3] used to formulate the main result i.e. Proposition 8. (The use of filtering functions makes sense though, if one relaxes the definition of a correlation function. However, in this paper we stick to the definition of a correlation function given in [3].)

In the modified algorithm, only correlation functions are computed in topological order of N_1 and N_2. The algorithm starts with block implementations $N_1(G_k),N_2(G_k)$ of level 1 then process implementations of level 2 and so on. Let *level*($N_1(G_k)$)=level($N_2(G_k)$)=1 (i.e. inputs of $N_1(G_k)$ and $N_2(G_k)$ are primary inputs of N_1 and N_2.) Let C be the variable associated with the output of G_k and $Cf(v_1(C),v_2(C))$ be the correlation function relating encodings $q_1(C)$ and $q_2(C)$. This function is obtained from the function $Sat(v(N_1(G_k)) \wedge Sat(v(N_2(G_k))$ by existentially quantifying away all the variables except the variables associated with the outputs of $N_1(G_k)$ and $N_2(G_k)$.

Suppose *level*($N_1(G_k)$)=*level*($N_2(G_k)$)=k and the correlation functions have been computed for the implementations of levels less than k. Let the output of G_k be associated with variable C_k. Let the inputs of G_k be connected to the outputs of blocks $G_{k1},\ldots,G_{km}$ associated with variables $C_{k1},..,C_{km}$ respectively. Then the correlation function $Cf(v_1(C_k),v_2(C_k))$ is obtained from the function $Cf(v_1(C_{k1}),v_2(C_{k1})) \wedge .. \wedge Cf(v_1(C_{km}),v_2(C_{km})) \wedge Sat(v(N_1(G_k)) \wedge Sat(v(N_2(G_k)))$ by existentially quantifying away all the variables except ones associated with the outputs of $N_1(G_k)$ and $N_2(G_k)$. Eventually the correlation function $Cf(f_1,f_2)$ is computed where f_1 and f_2 are Boolean variables specifying the outputs of N_1 and N_2. If $Cf(f_1,f_2) = (f_1 \vee \sim f_2) \wedge (\sim f_1 \vee f_2)$ (which is the equivalence function), then circuits N_1 and N_2 are functionally equivalent.

Definition 10. Given two functionally equivalent Boolean circuits N_1, N_2, S is called the ***finest common specification*** if it has the smallest granularity p among all the CSs of N_1 and N_2.

The complexity of the described algorithm is exponential in the granularity of S (i.e. in the size of the maximal subcircuit $N_i(G_k)$). So to evaluate the complexity of equivalence checking of N_1 and N_2 correctly, one needs to know the finest CS of N_1 and N_2 or a CS whose granularity is close to that of the finest one.

4. Toggle equivalence of Boolean functions

In this section, we introduce the notion of toggle equivalence. We also show that toggle equivalent Boolean functions can be considered as different implementations of the same multi-valued function.

Definition 11. Let $f_1:\{0,1\}^n \rightarrow \{0,1\}^m$ and f_2 $\{0,1\}^n \rightarrow \{0,1\}^k$ be m-output and k-output Boolean functions of the same set of variables. Functions f_1 and f_2 are called ***toggle equivalent*** if $f_1(x) \neq f_1(x') \Leftrightarrow f_2(x) \neq f_2(x')$. Circuits N_1 and N_2 implementing toggle equivalent functions f_1 and f_2 are called ***toggle equivalent circuits***.

Remark 3. Toggle equivalence means that for any pair of input vectors x, x' for which at least one output of f_1 "toggles", the same is true for f_2 and vice versa.

Definition 12. Let f be a multi-output Boolean function of n arguments. Denote by $Part(f)$ the partition of the set $\{0,1\}^n$ into blocks $B_1,\ldots,B_k$ such that $f(x) = f(x')$ if x, x' are in the same block and $f(x) \neq f(x')$ if x, x' are in different blocks.

Proposition 1. Let f_1 and f_2 be toggle equivalent. Then $Part(f_1)=Part(f_2)$ i.e. for each block B_i of $Part(f_1)$ there is a block B'_j of $Part(f_2)$ such that $B_i=B'_j$ and vice versa.

Proof. Assume the contrary i.e. $Part(f_1) \neq Part(f_2)$. Then there is a block B_i of $Part(f_1)$ such that no block B'_j of $Part(f_2)$ is equal to B_i. Then only the two cases below are possible.

a) B_i contains at least two input vectors. Then there is a pair of vectors x,x' of block B_i such that they are in different blocks of $Part(f_2)$. This means that $f_1(x)=f_1(x')$ while $f_2(x) \neq f_2(x')$ i.e f_1 and f_2 are not toggle equivalent. So we have a contradiction.

b) B_i contains only one input vector x. Let B'_j of $Part(f_2)$ contain vector x. Block B'_j also contains at least one more input vector because otherwise $B_i=B'_j$. So the block B'_j contains two input vectors that are in different blocks of $part(f_1)$. Hence f_1 and f_2 are not toggle equivalent and we again have a contradiction □

Proposition 2. Let f_1 and f_2 be toggle equivalent single output Boolean functions. Then $f_1=f_2$ or $f_1=\sim f_2$ where '~' means negation.

Proof. From Proposition 1 it follows that $Part(f_1)=Part(f_2)$. Since f_1, f_2 are Boolean functions, $Part(f_1)$ and $Part(f_2)$ contain two blocks each. So the only two alternatives are $f_1=f_2$ or $f_1=\sim f_2$.

Proposition 3. Let f_1 and f_2 be two multi-output Boolean functions of n Boolean variables such that $Part(f_1)$=Part(f_2). Then f_1 and f_2 are toggle equivalent.

Proof can be performed by reasoning in the same way as in the proof of Proposition 1.

Proposition 4. Let F be a multi-valued function of n Boolean variables. Let C be the multi-valued variable specifying the output of F. Let f_1 and f_2 be Boolean functions obtained from F by using encodings q_1 and q_2 (of possibly different length) for the values of C. Then f_1 and f_2 are toggle equivalent.

Proof. According to Definition 1 different values of C are assigned different codes. So $Part(f_1)=Part(f_2)=Part(F)$ where $Part(F)=\{B_1,\ldots,B_k\}$ and B_i consists of all the input vectors for which F takes the same value of C. From Proposition 3 it follows that f_1 and f_2 are toggle equivalent.

Proposition 5. Let f_1 and f_2 be toggle equivalent. Then f_1 and f_2 are two different "implementations" of the same multi-valued function of Boolean variables.

Proof According to Proposition 1 $Part(f_1)=Part(f_2)$. Let $Part(f_1)$,$Part(f_2)$ contain k blocks. Then f_1 and f_2 are implementations of the function F: $\{0,1\}^n \rightarrow \{1,..,k\}$ where $F(x)=m$, iff x is in the m-th block of $Part(f_1)$.

So far we have considered toggle equivalence of functions with identical sets of arguments. Below, the notion of toggle equivalence is extended to the case of Boolean circuits with different sets of arguments that are related by encoding functions.

Definition 13. Let $X=\{x_1,\ldots,x_n\}$ and $Y=\{y_1,\ldots,y_k\}$ be two disjoint sets of Boolean variables. Denote by $Enc(X,Y)$ a Boolean function satisfying the following two conditions

1). There do not exist three vectors x, x', y (where x, x' are assignments to variables X and y is an assignment to variables Y) such that $x \neq x'$ and $Enc(x, y)=Enc(x', y)=1$.

2) There do not exist three vectors x, y, y' such that $y \neq y'$ and $Enc(x,y)=Enc(x, y')=1$.

The function Enc is called an ***encoding function***.

Remark 4. An encoding function can be viewed as specifying two different encodings of the same multi-valued variable. Indeed, let $V=\{x_1,\ldots,x_k\}$ be the set of all assignments to variables of X such that $Enc(x_i, y)=1$ for some y. Let $W=\{y_1,.., y_m\}$ be the set of all assignments to variables of Y such that $Enc(x, y_i) = 1$ for some x. It is not hard to see that from Definition 13, it follows that $|W|=|V|$ and there exists a "natural" bijective mapping between X and Y that relates a vector x_i of V and the vector y_j of W for which $Enc(x_i, y_j)=1$. Vectors x_i and y_j can be considered as codes of the same value of a k-valued variable.

Definition 14. Let f_1:$\{0,1\}^n \rightarrow \{0,1\}^m$ and f_2 $\{0,1\}^p \rightarrow \{0,1\}^k$ be m-output and k-output Boolean functions and sets $X=\{x_1,\ldots,x_n\}$ and $Y=\{y_1,\ldots, y_p\}$ specify their arguments. Let $X=\{X_1,..,X_s\}$ and $Y=\{Y_1,\ldots,Y_s\}$ be partitions of X and Y into s blocks and $Enc(X_1,Y_1),\ldots, Enc(X_s,Y_s)$ are encoding functions. Functions f_1 and f_2 are called ***toggle equivalent under input encoding function*** $Enc(X,Y)=Enc(X_1,Y_1) \wedge \ldots \wedge Enc(X_s,Y_s)$ if $(f_1(x) \neq f_1(x') \wedge (Enc(x,y)=Enc(x', y')=1)) \Rightarrow (f_2(y) \neq f_2(y')$ and vice versa $(f_2(y) \neq f_2(y') \wedge (Enc(x,y)=Enc(x', y')=1)) \Rightarrow f_1(x) \neq f_1(x')$.

Proposition 6. Let f_1 and f_2 be toggle equivalent under input encoding function $Enc(X,Y)$=Enc(X_1,Y_1) $\wedge$ ... $Enc(X_s,Y_s)$. Then f_1 and f_2 are "implementations" of the same multi-valued function of s multi-valued arguments.

Proof follows from Proposition 5 and Remark 4.

5. Common specification and toggle equivalence

In this section, we show that the existence of a CS of circuits N_1 and N_2 means that N_1,N_2 can be partitioned into

toggle equivalent subcircuits that are connected in N_1 and N_2 "in the same way".

Definition 15. Let $N = (V,E)$ be a DAG representing a Boolean circuit (here V,E are sets of nodes and edges of N respectively.) A subgraph $N^*=(V^*, E^*)$ of N is called a ***subcircuit*** if the following two conditions hold:
a) if g_1,g_2 are in V^* and there is a path from g_1 to g_2 in N, then all the nodes of N that on that path are in V^* ;
b) if g_1,g_2 of V^* are connected by an edge in N, then they are also connected by an edge in N^*.

Definition 16. Let N^* be a subcircuit of N. An input of a gate g of N^* is called ***an input*** of N^* if it is not connected to the output of some other gate of N^*. A gate g of N^* is called an ***internal gate*** if all the gates of N whose inputs are fed by the output of g are in N^*. Otherwise, g is called ***an external gate***. The output of an external gate is called ***an output*** of circuit N^*.

Definition 17. Let N^* be a subcircuit of N of k inputs and p outputs. Let N^c be the circuit obtained from N by replacing N^* with a k-input, p-output node that inherits all the connections of subcircuit N^* to the gates of N that are not in N^*. We will say that N^c is obtained from N by ***collapsing*** the subcircuit N^*.

Definition 18. Let $\{N^1, ...,N^k\}$ be a partition of N into subcircuits. This partition is called ***topological*** if for each pair of subcircuits N^i,N^j the following condition holds: if there is node g' of N_i and node g'' of N_j that are connected by a path in N and $level(g') < level(g'')$, then for any pair of nodes g^* and g^{**} (where g* is in N^i and g^{**} is in N^j) it is true that $level(g^*) < level(g^{**})$ ($level(g)$ is the level of g in N).

Definition 19. Let N be a Boolean circuit and $N^1, ..., N^k$ be a topological partition of N into subcircuits. Let T be a directed graph obtained from the DAG of N using the following steps:
1) Each subcircuit N^i is collapsed in N (i.e. replaced with a node G_i with the same number of inputs and outputs as N^i).
2) The outputs of each node G_i are merged into one. If two (or more) outputs of G_i are connected to inputs of node G_m, then all the inputs of G_m connected to G_i but one are removed.
T is called ***the communication specification*** corresponding to the topological partition $N^1, ..., N^k$.

Remark 5. Informally, T describes information flow between subcircuits $N^1, ..., N^k$. The output of G_i is connected to an input of G_k in T iff an output of N^i is connected to an input of N^k in N.

Remark 6. It is not hard to show that since $N^1, ..., N^k$ is a topological partition, T is a DAG (i.e. T is acyclic).

Definition 20. Let T be the communication specification of circuit N with respect to a topological partition $N^1, ..., N^k$. Let G_i be the node of T corresponding to subcircuit N^i. The longest path from an input of T to G_i is called the level of G_i and N^i (denoted by $level(G_i)$ and $level(N^i)$ respectively).

Definition 21. Let $N_1^1, ...,N_1^k$ and $N_2^1, ..., N_2^k$ be topological partitions of single output Boolean circuits N_1,N_2. Let communication specifications of N_1 and N_2 with respect to partitions $N_1^1, ..., N_1^k$ and $N_2^1, ..., N_2^k$ be identical. Denote by $Cf(N_1^m,N_2^m)$, $m=1,...,k$ the correlation functions computed exactly as it was described in Section 3. Namely, we first compute correlation functions for subcircuits of level 1, then for level 2 and so on. The correlation function $Cf(N_1^m,N_2^m)$ is obtained from the function $H=Sat(v(N_1^m)) \wedge Sat(v(N_2^m)) \wedge Cf^*$, the function Cf^* being the conjunction of correlation functions for all the subcircuits N_1^i,N_2^i whose outputs are connected to inputs of N_1^m , N_2^m. The function $Cf(N_1^m,N_2^m)$ is obtained from H by existentially quantifying away all the variables except the output variables of N_1^m, N_2^m.

Proposition 7. Let f_1 and f_2 be two Boolean functions that are toggle equivalent under input encoding function $Enc(X,Y)=Enc(X_1,Y_1) \wedge ... \wedge Enc(X_s,Y_s)$ (see Definition 14.) Let N_1 and N_2 be circuits implementing f_1 and f_2 and V and W be the output variables of N_1 and N_2. Let $Cf(V,W)$ be the function obtained from the function $Enc(X,Y) \wedge Sat(v(N_1)) \wedge Sat(v(N_2))$ by existentially quantifying away all the variables except variables of V and W. Then $Cf(V,W)$ is an encoding function.

Proof. Assume the contrary i.e. $Cf(V,W)$ is not an encoding function. Suppose, for example, that there are vectors v,v',w such that $v \neq v'$ and $Cf(v,w)=Cf(v',w)$. Since Cf is obtained by existentially quantifying away some variables, there must exist vectors $z=(x,y,...,v,w)$ and $z' =(x',y',...,v',w)$ such that

1) input variables of N_1 and N_2 are assigned x, y in z and x', y' in z' and

2) output variables of N_1 and N_2 are assigned v,w in z and v',w' in z'.

Since $v \neq v'$, then $x \neq x'$. Hence there are vectors x,x',y,y' such that $x \neq x'$, $Enc(x,y)=Enc(x', y')=1$, $N_1(x)=v,N_1(x')=v'$ and $N_2(y)=w,N_2(y')=w$. But this means that N_1 and N_2 are toggle inequivalent and so we have a contradiction.

Proposition 8. Let $N_1^1, ..., N_1^k$ and $N_2^1, ..., N_2^k$ be topological partitions of functionally equivalent circuits N_1 and N_2 . Let communication specifications T_1 and T_2 of N_1 and N_2 with respect to these partitions be identical i.e. $T_1=T_2=T$. Let each pair of subcircuits N_1^m and N_2^m be toggle equivalent under input encoding function Cf^* (see Definition 21). Then N_1 and N_2 have a common specification S. The topology of S is specified by the communication specification T and N_1^m and N_2^m are implementations of m-th block of S.

Proof. By assumption each pair of circuits N_1^m, N_2^m are toggle equivalent if their inputs are restricted by the corresponding correlation functions. According to Proposition 6, if the correlation functions are encoding functions, then N_1^m and N_2^m implement the same multi-valued function. So one just needs to prove that all correlation functions are encoding functions. Let $level(N_1^m)=level(N_2^m)=1$. Then inputs of N_1^m and N_2^m are common primary inputs of N_1 and N_2 . One can view inputs of N_1^m and N_2^m as related by the encoding function that is the conjunction of functions describing equivalency of the corresponding input variables of N_1^m and N_2^m. According to Proposition 7, the correlation function relating outputs of N_1^m and N_2^m is an encoding function. Then by induction in topological levels one can easily prove that correlation function $Cf(N_1^p, N_2^p)$ is an encoding function for any value of p.

6. A method of HLLS

In previous sections we developed some theory that allows one to check the correctness of a CS of circuits N_1 and N_2 even if this specification is described implicitly. Suppose that N_1 and N_2 have the same topological specification defined by subcircuits N_1^1, ..., N_1^k and N_2^1, ..., N_2^k and corresponding pairs of subcircuits are toggle equivalent. Then these two sets of subcircuits give an implicit representation of a CS of N_1 and N_2. In this section, we use this result to formulate a method of High-Level structure aware Logic Synthesis (HLLS).

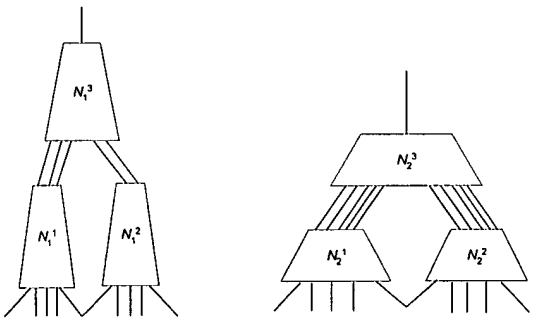

Figure 2. An example of HLLS

Let us describe this method by a simple example. Suppose circuit N_1 is partitioned into three subcircuits N_1^1, N_1^2, N_1^3 as shown in Figure 2 on the left. Suppose that we want to obtain a circuit N_2 with a better performance than N_1. This can be done by replacing "tall" subcircuits N_1^1, N_1^2, N_1^3 with "short" and "wide" subcircuits N_2^1, N_2^2, N_2^3 shown in Figure 2 on the right. In the method of HLLS this replacement is done in the following way. First we pick a subcircuit of topological level 1, say N_1^1 and synthesize a "short" subcircuit N_2^1 that is toggle equivalent to N_1^1. The same procedure applies to the other subcircuit of topological level 1. The subcircuit N_1^2 is replaced with a toggle equivalent subcircuit N_2^2. After we are done with level 1, we move to topological level 2 i.e. to the subcircuit N_1^3.

However, before the re-synthesis of circuit N_1^3 we need to compute the necessary correlation functions. The inputs of N_1^3 are connected to the outputs of circuits N_1^1 and N_1^2 , so we need to compute the correlation functions $Cf(N_1^1, N_1^2)$ and $Cf(N_1^2, N_2^2)$. This computation is done as described in Section 3. Then we build a subcircuit N_2^3 that is toggle equivalent to the subcircuit N_1^3. The toggle equivalence of N_1^3 and N_2^3 is computed not "globally" (in terms of primary inputs of N_1 and N_2) but locally in terms of inputs of N_1^3 and N_2^3 related by the correlation functions $Cf(N_1^1, N_2^1)$ and $Cf(N_1^2, N_2^2)$. Since subcircuits N_1^3 and N_2^3 have only one output, their toggle equivalence means that N_1^3 and N_2^3 (and hence circuits N_1 and N_2) are functionally equivalent modulo negation. Since N_1 and N_2 have the same topological specification and the corresponding subcircuits are toggle equivalent, N_2 is a different implementation of the same three-block specification that is implemented by N_1.

Of course, the described method gives only "the big picture" because the procedure for synthesis of toggle equivalent circuits is not described. The generalization of the method to a circuit N_1 with a specification of an arbitrary number of subcircuits is straightforward. The subcircuits of a specification of N_1 are replaced with toggle equivalent subcircuits in topological order. Toggle equivalence of subcircuits N_1^m and N_2^m is computed locally in terms of inputs of N_1^m and N_2^m related by correlation functions obtained before.

7. Relation of HLLS to other synthesis procedures

In this section, we discuss the relation between HLLS and existing synthesis procedures based on local transformations. The main three differences between such procedures and HLLS are shown in Table 1. The key idea of HLLS is to restrict the search to implementations of a predefined specification. The justification of such an approach is as follows. Suppose we need to optimize a combinational circuit N_1 that has a high-level specification S represented as a partition of N_1 into subcircuits N_1^1, ..., N_1^k. One can view these subcircuits as implementations of multi-valued blocks G_1,..,G_k of S obtained by encoding values of variables of S with binary codes (however it does not mean that this encoding was performed explicitly).

It is quite possible that the chosen codes are far from optimal and so a much better implementation N_2 of S than

N_1 may exist. However, a procedure based on local transformations, be it don't care based optimization [5] or SPFDS [6][7], will not be able to find N_2. This is because to produce a different implementation of S one has to perturb many nodes of N_1 at once (when replacing a subcircuit N_1^m with its toggle equivalent counterpart). On the other hand, it is unlikely that by local transformations one can produce a circuit (that is not an implementation of S) "better" than N_2.

Table 1. Comparison of HLLS and procedures based on local transformations

Synthesis based on local transformations	*HLLS*
Circuit is optimized node by node	Many nodes are perturbed "at once"
Equivalence checking is not an issue	Reducing complexity of equivalence checking is crucial
The search space is not restricted	One considers only implementations of a predefined specification.

Of course, the conjecture that the quality of encodings matters so much is based on the assumption that the specification S captures essential features of the circuit to be implemented. An example, of a circuit with a "meaningful" specification is a multiplier. On the other hand, if N_1 has many specifications, sticking to one of them does not make much sense.

It is not hard to see that HLLS and synthesis procedures based on local transformations are complementary and so can be used together. An HLLS procedure can be used as a first step of logic optimization meant to improve encodings of multi-valued variables and so to generate a better starting point for a procedure based on local transformations.

8. A procedure for testing toggle equivalence

Checking toggle equivalence is a key operation of the method of HLLS described in Section 6. In this section, we describe a procedure for testing toggle equivalence and give some experimental results. Let N_1 and N_2 be two multi-output Boolean circuits. Denote by $inp(N_1)$,$inp(N_2)$ and $out(N_1)$,$out(N_2)$ the set of primary input and output variables of circuits N_1 and N_2. Denote by $Enc(inp(N_1),inp(N_2))$ a function specifying allowable combinations of inputs for N_1 and N_2. Let N^*_1 and N^*_2 be copies of circuits N_1 and N_2 respectively that depend on different sets of variables (shown in Fig. 2.) Denote by H the function $Sat(v(N_1)) \wedge Sat(v(N_2)) \wedge Sat(v(N^*_1)) \wedge Sat(v(N^*_2)) \wedge Enc(inp(N_1),inp(N_2)) \wedge Enc(inp(N^*_1),inp(N^*_2))$. Denote by $Eq(out(N_1),\text{out}(N^*_1))$ the function that is equal to 1 iff $out(N_1)$ and $out(N^*_1)$ take the same value. Denote by $Neq(out(N_1),out(N^*_1))$ the function that is equal to 1 iff $out(N_1)$ and $out(N^*_1)$ take different values. Denote by H_1 the function $H \wedge Eq(out(N_1),out(N^*_1)) \wedge Neq(out(N_2),out(N^*_2))$. Denote by H_2 the function $H \wedge Neq(out(N_1),\text{out}(N^*_1)) \wedge Eq(out(N_2),\text{out}(N^*_2))$.

Proposition 9. Circuits N_1 and N_2 are toggle equivalent iff the functions H_1 and H_2 (defined above) are constant 0.

If part. Assume the contrary i.e. H_1 and H_2 are constants 0 and N_1 and N_2 are not toggle equivalent. Suppose there are vectors $(\boldsymbol{x},\boldsymbol{h})$ and $(\boldsymbol{x}^*,\boldsymbol{h}^*)$ such that $N_1(\boldsymbol{x}) = N_1(\boldsymbol{x}^*)$ while $N_2(\boldsymbol{h}) \neq N_2(\boldsymbol{h}^*)$ where $Enc(\boldsymbol{x},\boldsymbol{h})=Enc(\boldsymbol{x}^*,\boldsymbol{h}^*)=1$. Then there is a vector $\boldsymbol{w}$ that satisfies the function H_1 (and so we have a contradiction). Indeed, let $\boldsymbol{w}$ be the set of assignments to the variables of circuits N_1,N_2,N_1^*,N^*_2 under the input assignment $(\boldsymbol{x},\boldsymbol{h},\boldsymbol{x}^*,\boldsymbol{h}^*)$. The assignment $\boldsymbol{w}$ satisfies $Eq(out(N_1),out(N^*_1))$ because $N_1(\boldsymbol{x}) = N_1(\boldsymbol{x}^*)$. It also satisfies $Neq(out(N_2),out(N^*_2))$, because $N_2(\boldsymbol{h}) \neq N_2(\boldsymbol{h}^*)$. Besides, $\boldsymbol{w}$ satisfies function H (because $Enc(\boldsymbol{x},\boldsymbol{h})=Enc(\boldsymbol{x}^*,\boldsymbol{h}^*)=1$ and $\boldsymbol{w}$ is picked in such a way that it satisfies $Sat(v(N_1))$, $Sat(v(N_2))$, $Sat(v(N^*_1))$, $Sat(v(N^*_2))$.

Only if part. Assume the contrary i.e. circuits N_1 and N_2 (and hence N^*_1 and N^*_2) are toggle equivalent but either H_1 or H_2 (or both) is satisfiable for some assignment of values. Suppose for example that there is an assignment $\boldsymbol{w}$ such that $H_1(\boldsymbol{w})=1$. Let $\boldsymbol{x}$, $\boldsymbol{h}$, $\boldsymbol{x}^*,\boldsymbol{h}^*$ be the parts of vector $\boldsymbol{w}$ that are assignments to the input variables of N_1,N_2,N^*_1,N^*_2 respectively. Since vector $\boldsymbol{w}$ satisfies $Sat(v(N_1))$, $Sat(v(N_2))$, $Sat(v(N^*_1))$, $Sat(v(N^*_2))$., then the rest of the assignments in $\boldsymbol{w}$ are consistent with $\boldsymbol{x},\boldsymbol{h}$, $\boldsymbol{x}^*,\boldsymbol{h}^*$ i.e. they are the values of gates of N_1,N_2,N^*_1,N^*_2 under the input vector $\boldsymbol{x},\boldsymbol{h},\boldsymbol{x}^*,\boldsymbol{h}^*$ respectively. Besides, since $Enc(\boldsymbol{x},\boldsymbol{h})= Enc(\boldsymbol{x}^*,\boldsymbol{h}^*)=Eq(out(N_1),out(N^*_1))\wedge Neq(out(N_2),out(N^*_2))=1$, then $N_1(\boldsymbol{x}) = N_1(\boldsymbol{x}^*)$ while $N_2(\boldsymbol{h}) \neq N_2(\boldsymbol{h}^*)$. This means that N_1 and N_2 are not toggle equivalent and so we have a contradiction.

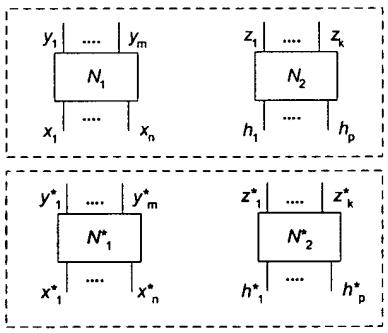

Figure 3 Two copies of circuits N_1 and N_2 to be checked for toggle equivalence

In the experiments we used 39 circuits from the MCNC benchmark set. Namely, for each circuit N_1 listed in Table 2 we checked its toggle equivalence with a circuit N_2 produced from N_1 by a random permutation of outputs. Clearly, permutation of outputs destroys functional equivalence of N_1 and N_2 but preserves their toggle equivalence. To check if N_1 and N_2 are toggle equivalent

Table 2. Toggle equivalence checking for MCNC circuits

name	#inputs	#outputs	time (sec.)
pcler8	27	17	0.01
frg1	28	3	0.03
sct	19	15	0.04
unreg	36	16	0.04
lal	26	19	0.05
c8	28	18	0.07
cht	47	36	0.08
b9	41	21	0.09
my_adder	33	17	0.16
example2	85	66	0.17
C432	36	7	0.18
apex7	49	37	0.18
vda	17	39	0.18
ttt2	24	21	0.33
i5	133	66	0.40
i6	138	67	0.63
term1	34	10	0.74
i7	199	67	0.99
i9	88	63	0.99
K2	45	43	1.46
apex6	135	99	1.58
x4	94	71	1.58
x3	135	99	1.70
x1	51	35	2.28
C499	41	32	2.41
rot	135	107	2.97
C880	60	26	5.07
frg2	143	139	5.13
C1355	41	32	6.48
pair	173	137	9.22
des	256	245	40.39
C1908	33	25	47.10
too_large	38	3	70.9
i8	133	81	150.02
C5315	178	123	193.10
C3540	50	22	261.28
dalu	75	16	310.67
i10	257	224	588.87
C7552	207	108	6,122.5
Geometric mean			1.84
Arithmetic mean			200.77

we created CNFs describing functions H_1 and H_2 and checked them for satisfiability. (Since tested pairs N_1,N_2 were toggle equivalent, all generated CNF formulas were unsatisfiable.) For satisfiability testing we used the SAT-solver BerkMin[1],[2]. The runtimes are shown in the last column of Table 2. It is not hard to see that in the majority of cases, toggle equivalence was established very quickly which proves that the proposed procedure may be used in HLLS.

9. Conclusions

We introduce a method of High-Level structure aware Logic Synthesis (HLLS). The key idea of the method is to re-encode multi-valued variables of the specification describing the high-level structure of the circuit implicitly at the gate level. We show that this can be done by preserving toggle equivalence among initial and re-synthesized implementations of multi-valued blocks. We show that toggle equivalence can be efficiently checked for relatively large pieces of combinational logic, which makes HLLS a promising direction for future research.

References

[1] BerkMin web page. *http://eigold.tripod.com/BerkMin.html*

[2] Goldberg E.,Novikov,Y. *BerkMin: A fast and robust SAT-solver.* Design, Automation, and Test in Europe (DATE '02), pages 142-149, March 2002.

[3] E.Goldberg, Y. Novikov. *Equivalence Checking of Dissimilar Circuits.* International Workshop on Logic and Synthesis, May 28-30, 2003, USA. Available at *http://eigold.tripod.com/papers/dissim-iwls.zip*

[4] E.Goldberg, Y.Novikov. *How good can a resolution based SAT-solver be?* In the proceedings of 6-th International Conference on Theory and Applications of Satisfiability Testing, Italy, 2003,LNCS 2919,pp.37-52.

[5] H.Savoj. Don't cares in multi-level network optimization. Ph.D. thesis, UC Berkeley, 1992.

[6] S.Sinha, R.Brayton. Implementation and use of SPFDs in optimizing Boolean networks. ICCAD,pp.103-110,1998.

[7] S.Yamashita, H.Sawada, and A.Nagoya. A New method to express functional permissibilities for LUT based FPGAs and its applications. ICCAD, pp. 254-261, 1996.

Wireplanning in Logic Synthesis *

Wilsin Gosti Amit Narayan+ Robert K. Brayton Alberto L. Sangiovanni-Vincentelli
Dept. of EECS, University of California, Berkeley, CA 94720
+ Monterey Design Systems, Sunnyvale, CA 95139
Email: {wilsin, anarayan, brayton, alberto}@eecs.berkeley.edu

Abstract

In this paper, we propose a new logic synthesis methodology to deal with the increasing importance of the interconnect delay in deep-submicron technologies. We first show that conventional logic synthesis techniques can produce circuits which will have long paths even if placed optimally. Then, we characterize the conditions under which this can happen and propose logic synthesis techniques which produce circuits which are "better" for placement. Our proposed approach still separates logic synthesis from physical design.

1 Introduction

Conventional logic synthesis assumes that the delay of a circuit depends only on the delays of the gates in the circuit and mostly ignores the effect of interconnect delay. However, as we move towards smaller geometries, interconnect delay is becoming an increasingly larger fraction of the total delay. In fact, Semiconductor Industrial Alliance's National Technology Roadmap for Semiconductor for 1997 [1] predicts that interconnect delay will start dominating the total gate delay as we move down to 0.15μ technology and below. Another study by Keutzer, et al [5] shows that for 0.25μ technology and below, interconnect delay can contribute anywhere from 50% to 80% of the total delay. Therefore, logic synthesis can no longer afford to ignore the effect of interconnect delay during optimization.

In this paper, we adopt a diametrically opposite approach to that of conventional logic synthesis. We perform logic synthesis to optimize only for interconnect delay, ignoring the effect of gate delays. Our approach is based on the simple observation that if an output o depends on an input i, then the best way to connect i to o is through a path which is monotonic from i to o, that is, there are no diversions in the path from i to o. We first show, by means of an example, that conventional logic synthesis can produce a circuit for which it is impossible to find a placement with no diversions in the input-output paths. Therefore, no matter how good a place & route tool is, it may not be able to produce a circuit which is optimal in terms of interconnect delay.

We define the notion of *illegal* nodes. Intuitively speaking, a node is illegal if it introduces a diversion in the circuit no matter where it is placed. We characterize the condition under which a node is illegal and give a procedure to convert an arbitrary circuit into a circuit which has only legal nodes. We call such a circuit a *legal* circuit. We show that for a legal circuit, there always exists a point placement of the nodes such that every input-output path is monotonic. We also provide a set of logic synthesis transformations which are guaranteed to preserve the "legality" of a circuit.

The proposed approach has the advantage that it still maintains a distinction between the logic synthesis and place & route stages. It does not need to tightly couple synthesis and placement by frequently alternating between the two which can be inefficient and may not converge at all.

2 Previous Work

So far very little work has been done to model the effect of interconnect delay at the logic level. This is mainly due to the fact that at the logic level, very little information is available about the interconnect. Most of these approaches [9, 8, 14] use a rough companion placement to estimate the cost of various logic synthesis operations and make decisions based on this cost. In [13] an iterative approach to combine synthesis and placement is presented. Instead of using a companion placement to guide synthesis, they use actual placement which can be modified incrementally based on the netlist changes. In [15] a heuristic to minimize the layout cost is proposed which doesn't employ a companion placement solution. The method in [15] is based on minimizing the average fanout range and evenly distributing fanouts in the chip. It was shown that the chip delay could be reduced by this approach if all the input pins are located on one side of the chip and all the output pins on the opposite. Like [15], our approach also does not employ a companion placement. We analyze conditions under which a netlist is not "good" for placement given the locations of i/o pins and try to transform it into one which is.

3 Preliminaries

Definition 1 *A logic circuit $\mathcal{L}$ is a 3-tuple $(I, O, \mathcal{F})$. I is a set of primary input pins or simply primary inputs. O is a set of primary output pins or simply primary outputs. Each element of I and O is a binary variable. An element $f_j \in \mathcal{F}$ is a function $f_j : \mathbf{B}^{|I|} \mapsto \mathbf{B}$. Each f_j is called the global function of the primary output o_j.*

A logic circuit is represented by a Boolean network [3]. If n_k is an immediate fanout of n_j in the Boolean network, we write $n_j \rightarrow n_k$. A logic circuit is *pin-assigned* if each primary input i is labeled with a position (x_i, y_i) and each primary output o with position (x_o, y_o). A logic circuit $\mathcal{L}$ is *placed* if every node n of the Boolean network representing $\mathcal{L}$ has a position, i.e. every node n is labeled with (x_n, y_n), and the resulting placement is denoted by $\mathcal{P}_{\mathcal{L}}$. A point placement of $\mathcal{L}$ is a placement of $\mathcal{L}$ where each node is represented as a point. Given a point-placed circuit, a path, $p_{(i,o)}$, from a primary input i to a primary output o is a sequence of connected nodes from i to o, and the length of the path, $d_{(i,o)}$, is the length of all the wires along the path from i to o. The path $p_{(i,o)}$ is called *monotonic* if its length is equal to the

*This work was supported in part by SRC-98-DC-324.

ICCAD98, San Jose, CA, USA
© 1998 ACM 1-58113-008-2/98/0011_$5.00

Manhattan distance from i to o. The placement $\mathcal{P}_{\mathcal{L}}$ of $\mathcal{L}$ is optimal if there is no other placement of $\mathcal{L}$ whose length of the longest path is shorter than that of $\mathcal{P}_{\mathcal{L}}$.

The coordinate system that we use in the paper assumes that the x-axis goes from left to right and the y-axis goes from top to bottom.

4 Problem Description

Given a logic circuit $\mathcal{L}$, the goal is to find a placed circuit $\mathcal{N}_{\mathcal{L}}$ such that the interconnect delay of the circuit is minimized. Due to efficiency reasons, we want to maintain the decoupling of the problem into a separate synthesis phase followed by a place & route phase as in the conventional approach. Given a logic circuit $\mathcal{L} = (I, O, \mathcal{F})$, we address the problem of finding a Boolean network $\mathcal{N}_{\mathcal{L}}$, which when placed optimally, leads to a circuit with minimum interconnect delay. It is up to the placement tool to find the optimal placement for such a network. Intuitively speaking, we are trying to create a circuit for which a "good" placement exists.

We assume that the die is represented by a rectangle R with width w_R and height h_R and the given logic circuit is pin-assigned. We assume that the delay of a path is a linear function of its length. In general, the interconnect delay depends quadratically on the length of the interconnect. However, it can be made linear by buffer insertion and wire sizing, as shown in recent studies by Otten and Brayton [7] and Cong and Pan [4]. A circuit is said to be optimal in terms of interconnect delay if the length of a path from any primary input i to any primary output o is its Manhattan distance (monotonic), i.e.

$$d_{(i,o)} = |x_i - x_o| + |y_i - y_o|$$

This definition is motivated by the pin-to-pin delay model of Kukimoto and Brayton [6]. Under this model, a delay number is assigned for every input-output pair. This model is particularly suited for intellectual property (IP) blocks where the arrival time of the pins are not known in advance. Consequently, any input-output path can end up being a critical path. Therefore, to minimize the delay, we have to minimize the delay for all input-output paths. We call this problem the *IP-based synthesis* problem.

We will also be addressing a slightly different problem called the *slack-based synthesis* problem, where the only difference from the IP-based problem is the objective function. Instead of minimizing the length of the path from any primary input to any primary output, we minimize the longest path of the circuit, i.e.

$$\min\{\max_{(i \in I, o \in O)} d_{(i,o)}\}$$

In this paper, we will mainly focus on the *IP-based synthesis* problem. However, the approach can be modified to address the *slack-based* problem as well. We will very briefly discuss this in Section 7.

5 Approach

To understand the problem better, let us first look at an example where the conventional logic synthesis which considers only gates during optimization may not be able to find a circuit with minimum interconnect delay.

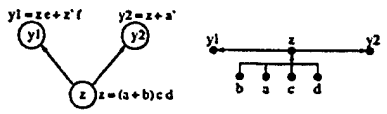

Figure 1: Network $\mathcal{N}_{\min}$ and its optimal placement.

5.1 Logic Synthesis and Interconnect Delay: An Example

Let us consider a minimum literal boolean network $\mathcal{N}_{\min}$ with 10 literals as shown in Figure 1 on the left. Assuming that the pin positions are given, the optimal placement of $\mathcal{N}_{\min}$ is shown in Figure 1 on the right. Pins e and f are not shown and are assumed to be close to y_1. In this solution, there are two longest paths of equal length, i.e. one path from b to y_1 and the other from b to y_2. This circuit is not optimal in terms of both the IP-based and the slack-based synthesis problems because there is a better decomposition of the circuit that produces shorter longest paths. The better decomposed network with 11 literals is shown in Figure 2 together with its optimal placement. Although network $\mathcal{N}_{\min}$ has fewer literals than $\mathcal{N}'$, it has an extra path from b to y_2. Consequently, the placement tool places node z to minimize the longest paths from b to y_1 and y_2. However, as we see in Figure 2, y_2 is independent of b and therefore, b can be removed from the support of y_2. This leads to the network in Figure 2 whose optimal placement has shorter longest path as compared to $\mathcal{N}_{\min}$.

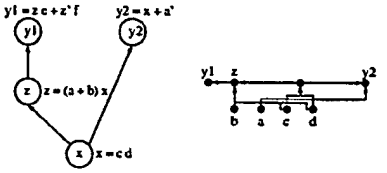

Figure 2: Network $\mathcal{N}'$ and its optimal placement.

Although network $\mathcal{N}'$ is better than $\mathcal{N}_{\min}$ in terms of both IP-based and slack-based synthesis problems, there is yet a better decomposition for the IP-based synthesis problem. In $\mathcal{N}'$, the path from c to y_1 is greater than its Manhattan distance. The same is true for the path from d to y_2. A better decomposed circuit $\mathcal{N}''$ with 11 literals and its optimal placement are shown in Figure 3.

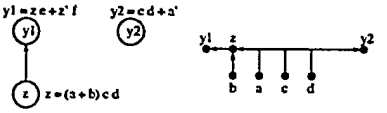

Figure 3: Network $\mathcal{N}''$ and its optimal placement.

From the example above, we see that sometimes the output of a logic synthesis is not "good" for placement, i.e. no matter how we place the nodes, there is at least one path which is longer than its Manhattan distance. In our approach, the aim is to guide logic synthesis such that it produces a circuit which is good for placement. It is up to the placement tool to find the optimal placement for the decomposed circuit in the placement phase.

In this section we define what we mean by a circuit which is "good" for placement and then give a set of transformation rules which can find such a circuit. Our approach can be divided into two broad stages: constraint generation and constraint driven synthesis. In the constraint generation step, we partition the die into regions and identify the types

of functions that are allowed to fill them. We define the notion of *illegal* nodes. Intuitively speaking, a node is illegal if it can not be placed somewhere on the die without causing a diversion in the circuit. We show that if a circuit consists of only legal nodes then there exists a point placement of the nodes such that every input-output path is monotonic. We call such a circuit a *legal* circuit. We characterize the condition under which a node is illegal and give a procedure to convert an arbitrary circuit into a legal circuit.

Since nodes have areas, in the constraint driven synthesis step, we synthesize the legal circuit to find another legal circuit with minimum area. We extend the algebraic transformations and don't care minimization such that they operate on legal nodes and produce legal nodes. As in the conventional logic synthesis case, we use the number of factored-form literals as our area estimates since it has been proven to be a good indication of the size of a Boolean network.

5.2 Constraint Generation

Since the length of every path from a primary input to a primary output is restricted to its Manhattan distance (monotonic), there is a well defined region where a Boolean node can be placed. Let us define region formally.

Definition 2 *A region* $r = \{x_l, y_t, x_r, y_b\}$, *where* $x_l \le x_r$ *and* $y_t \le y_b$, *is the set of all points in the rectangle bounded at opposite corners by the points* (x_l, y_t) *and* (x_r, y_b). *Mathematically,* $r = \{(x,y) \mid x_l \le x \le x_r \text{ and } y_t \le y \le y_b\}$.

Definition 3 *Given two points* $p_1 = (x_1, y_1)$ *and* $p_2 = (x_2, y_2)$, *the region defined by* p_1 *and* p_2 *is region* $r_{(p_1,p_2)} = \{\min(x_{p_1}, x_{p_2}), \min(y_{p_1}, y_{p_2}), \max(x_{p_1}, x_{p_2}), \max(y_{p_1}, y_{p_2})\}$.

With these definitions, we go back to analyze why node z of the Boolean network $\mathcal{N}'$ in Figure 2 is "good" but not x. Because node z fans out to y_1 and its support set is $\{a,b,c,d\}$, z should be placed in the region $r_{(y_1,b)}$, which is r_z in Figure 4, so that the path from any primary input in the support set, i.e. a, b, c, or d, to y_1 is monotonic. For the same example, there is no good region to place node x because there are two conflicting requirements. One requirement says that node x should be placed in region $r_{(y_1,c)}$, which is r_1 in Figure 5, for the path from c to y_1 to be monotonic; while the other says that node x should be placed in region $r_{(y_2,d)}$, which is r_2 in Figure 5, for the path from d to y_2 to be monotonic. As shown in the figure, x can not be placed in both r_1 and r_2. Hence, x is not a desirable factor.

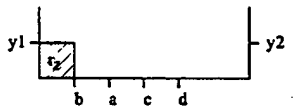

Figure 4: Legal region of node z.

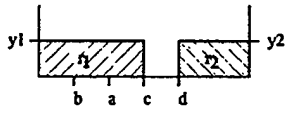

Figure 5: Conflicting legal region requirements for x.

5.2.1 Region Placement Constraints

The example above illustrates that if there is a path from a primary input i to a primary output o, then for the path to be monotonic, all the logic gates along the path should be placed in the region $r_{(i,o)}$. This leads us to first partition the die into rectangles along the pin positions and label each region with functions that can be placed in it. Continuing with our example, the die area associated with y_1, y_2, a, b, c, and d is partitioned into regions $\mathcal{R} = \{r_1, r_2, r_3, r_4, r_5\}$ as shown in Figure 6. Region r_1 is labeled with $\{a,b,c,d\}_{y_1}$ to mean that factors whose supports are a subset of $\{a,b,c,d\}$ and transitively fan out only to y_1 are allowed to be placed in r_1. Region r_3 is labeled with $\{c,d\}_{y_1}$ and $\{a,b\}_{y_2}$ to mean that factors whose supports are a subset of $\{c,d\}$ and transitively fan out only to y_1 or factors whose supports are a subset of $\{a,b\}$ and transitively fans out only to y_2 are allowed to be placed in r_3. Other regions are labeled in a similar fashion. Refering back to Boolean network $\mathcal{N}'$, we see that node z is a "good" node and can be placed in r_1 because its support set is $\{a,b,c,d\}$ and it transitively fans out only to y_1. This matches the label of r_1. Node x is not a "good" node because there is no region whose label contains its support set $\{c,d\}$ and both of its transitive fanouts are y_1 and y_2.

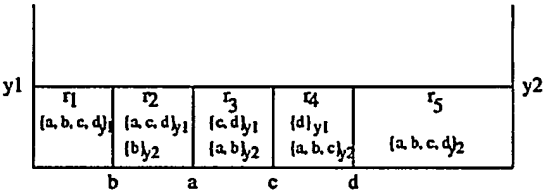

Figure 6: Regions and labels of regions.

Definition 4 *A placement constraint* d *is a 2-tuple* (O^d, σ^d), *where* $O^d \subseteq O$, *and* $\sigma^d \subseteq I$. O^d *is called the output set and* σ^d *the support set of* d. *We also write* d *as* $\{i_1, i_2, \ldots\}_{o_1,o_2,\ldots}$, *where* $\sigma^d = \{i_1, i_2, \ldots\}$ *and* $O^d = \{o_1, o_2, \ldots\}$.

Each region is labeled with a set of placement constraints, e.g. r_1 is labeled with $\{a,b,c,d\}_{y_1}$ and r_3 is labeled with $\{c,d\}_{y_1}$ and $\{a,b\}_{y_2}$ as shown in Figure 6. A placement constraint on a region r is called its *region placement constraint.*

Hence, each region placement constraint $d_r = (O^r, \sigma^r)$ in a region r denotes that Boolean nodes that fan out only to a subset of the primary outputs in O^r and have at most σ^r in their support can be placed in r.

5.2.2 Node Placement Constraints

We see that given a region r, only certain types of nodes can be placed in r and this is captured in its *region placement constraint.* We now define the dual for nodes. Given a node n, it can only be placed in certain regions. For example, node z of Boolean network $\mathcal{N}'$ in Figure 2 can only be placed in region r_1 as shown in Figure 6. Hence, we label each node with a placement constraint and it is called its *node placement constraint.* The node placement constraint of node n denotes the support of n and its transitive primary outputs. For example, the node placement constraint of z of Boolean network $\mathcal{N}'$ is $\{a,b,c,d\}_{y_1}$.

The node placement constraints of nodes of a Boolean network can be easily computed by traversing the Boolean network in a breadth-first manner from the primary inputs to compute the support sets and from the primary outputs to compute the output sets.

5.2.3 Properties of Placement Constraints on Boolean Networks

In this section, we show what "good" nodes mean and having a Boolean Network with only "good" nodes can lead to a monotonic point placement of the network.

Intuitively, a "good" node is one that can be placed in a region. We define such "good" nodes as legal. However, before we can formally define the legality of a node, we need the definition of containment of placement constraints.

Definition 5 *Placement constraint $d_a = (O^a, \sigma^a)$ is contained in placement constraint $d_b = (O^b, \sigma^b)$, denoted as $d_a \subseteq d_b$, if $O^a \subseteq O^b$ and $\sigma^a \subseteq \sigma^b$.*

Definition 6 *Boolean node n with node placement constraint d_n is* **legal** *with respect to region r with region placement constraints $\{d_{r_1}, d_{r_2}, \ldots\}$, denoted as $n \downarrow r$, if there exists a j such that $d_n \subseteq d_{r_j}$.*

Definition 6 says that node n is legal with respect to region r if n can be placed in r.

Definition 7 *A Boolean node n is* **legal** *if there is a region r such that $n \downarrow r$.*

Definition 7 says that node n is legal if there is a region r where n can be placed. This definition and Definition 6 are about the legality of a Boolean node. Now given a node, the next definition defines the region in which the node is legal.

Definition 8 *The legal regions of a node n, denoted as $R(n)$, is the set of regions $\mathcal{R} = \{r_1, r_2, \ldots, r_l\}$ such that for any region $r_j \in \mathcal{R}$, $n \downarrow r_j$.*

For clarity purposes, we denote the legal region of a node n with node placement constraint d_n as $R(d_n)$. We will then assume that given a node placement constraint, the node is implicitly defined.

It can be easily seen that $R(\{i_k\}_{o_l})$ is the region $r_{(i_k,o_l)}$. If we define $R(d_1) \cap R(d_2)$ to be the overlapping region between $R(d_1)$ and $R(d_2)$, then it is easy to see that $R(\{i_1, i_2, \ldots, i_m\}_{o_1,o_2,\ldots,o_n})$ is equal to:

$$\begin{aligned} &R(\{i_1\}_{o_1}) \cap R(\{i_2\}_{o_1}) \cap \cdots \cap R(\{i_m\}_{o_1}) \cap \\ &R(\{i_1\}_{o_2}) \cap R(\{i_2\}_{o_2}) \cap \cdots \cap R(\{i_m\}_{o_2}) \cap \\ &\qquad\qquad \cdots \cap \\ &R(\{i_1\}_{o_n}) \cap R(\{i_2\}_{o_n}) \cap \cdots \cap R(\{i_m\}_{o_n}). \end{aligned}$$

This is called the *intersection rule*. For example, as shown in Figure 7, for node z of Boolean network $\mathcal{N}'$,

$$\begin{aligned} R(z) &= R(\{a,b,c,d\}_{y_1}) \\ &= R(\{a\}_{y_1}) \cap R(\{b\}_{y_1}) \cap R(\{c\}_{y_1}) \cap R(\{d\}_{y_1}) \\ &= r_{z_1} \cap r_{z_2} \cap r_{z_3} \cap r_{z_4} \\ &= r_z \end{aligned}$$

Based on Definition 7, the legality of a node n with node placement constraint $d_n = (O^n, \sigma^n)$ can be checked by traversing all regions and check if n is legal for each region. Assuming $|I| > |O|$, the complexity for this algorithm is $O(|I|^2\,|O|)$ because the number of regions is $O(|I|\,|O|)$ and the number of region placement constraints in a region is $O(|I| + |O|)$. A better algorithm would be to check if the legal region of n is empty or not. This can be done by using the intersection rule defined above. The complexity is then $O(|O^n|\,|\sigma^n|)$, which can be much smaller. However, there is a linear algorithm with complexity $O(|O^n| + |\sigma^n|)$ according to the next three lemmas.

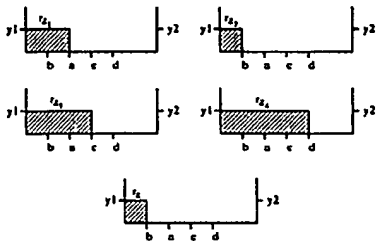

Figure 7: Region intersection for node z of $\mathcal{N}'$.

Lemma 1 below says that nodes that transitively fan out to only one output are always legal.

Lemma 1 *For a node placement constraint $\{i_1, i_2, \ldots, i_m\}_{o_1,o_2,\ldots,o_n}$ with $n = 1$, $R(\{i_1, i_2, \ldots, i_m\}_{o_1,o_2,\ldots,o_n}) \neq 0$.*

Proof: For $\{i_1, i_2, \ldots, i_m\}_{o_l}$, the point (x_{o_l}, y_{o_l}) is in $R(\{i_1, i_2, \ldots, i_m\}_{o_l})$. ■

Lemma 2 below enumerates the cases when nodes that transitively fan out to two outputs are legal.

Lemma 2 *For a node placement constraint $\{i_1, i_2, \ldots, i_m\}_{o_1,o_2,\ldots,o_n}$ with $m \geq 2$ and $n = 2$, $R(\{i_1, i_2, \ldots, i_m\}_{o_1,o_2,\ldots,o_n}) \neq 0$ iff*

1. *$(\forall i \forall o\, x_i \geq x_o \wedge y_i \geq y_o) \vee (\forall i \forall o\, x_i \geq x_o \wedge y_i \leq y_o) \vee (\forall i \forall o\, x_i \leq x_o \wedge y_i \geq y_o) \vee (\forall i \forall o\, x_i \leq x_o \wedge y_i \leq y_o)$, or*
2. *$R(\{i_1, i_2, \ldots, i_m\}_{o_l})$ is a point, i.e. $x_{o_1} = x_{o_2} \wedge \forall i\, y_i = C$, or $y_{o_1} = y_{o_2} \wedge \forall i\, x_i = C$, for some $C \in \mathcal{N}$.*

Proof: If part:

1. Let us assume without loss of generality that $(\forall i \forall o\, x_i \geq x_o \wedge y_i \geq y_o)$, and let $i_{\min} = (\min\{x_i\}, \min\{y_i\})$ and $o_{\max} = (\max\{x_o\}, \max\{y_o\})$, then the legal region is $r_{(i_{\min}, o_{\max})}$ and it is not empty.
2. If the legal region is a point, then it is not empty.

Only if part: Without loss of generality assume that the legal region is not empty and it is not a point, but $x_{i_1} < x_{o_1} < x_{i_2}$, i.e. o_1 is on the top side of the die, then $R(\{i_1, i_2\}_{o_1})$ is a point if both i_1 and i_2 are on the top side as well (Figure 8a); it is a line otherwise (Figure 8b). Since $R(\{i_1, i_2\}_{o_1,o_2}) = R(\{i_1, i_2\}_{o_1}) \wedge R(\{i_1, i_2\}_{o_2})$, it is not empty iff $y_{i_1} = y_{i_2}$ and $x_{o_1} = x_{o_2}$, i.e. o_1 and o_2 are at opposite side (Figure 8c). If we have more than two inputs, then they all have to be either on the top or the bottom side of the chip for the legal region to be non-empty and the legal region has to be a point (Figure 8d). Hence, it is a contradiction. ■

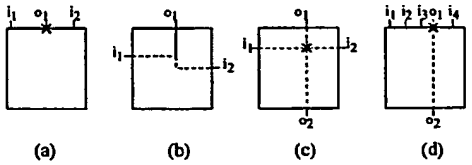

Figure 8: Figure for proof of Lemma 2.

The following lemma says if a node transitively fans out to more than two outputs, then there can only be one case where it is legal.

Lemma 3 *For a node placement constraint* $\{i_1, i_2, \ldots, i_m\}_{o_1,o_2,\ldots,o_n}$ *with* $m \geq 2$, *and* $n > 2$, $R(\{i_1, i_2, \ldots, i_m\}_{o_1,o_2,\ldots,o_n}) \neq 0$ *iff* $(\forall i \forall o\, x_i \geq x_o \wedge y_i \geq y_o) \vee (\forall i \forall o\, x_i \geq x_o \wedge y_i \leq y_o) \vee (\forall i \forall o\, x_i \leq x_o \wedge y_i \geq y_o) \vee (\forall i \forall o\, x_i \leq x_o \wedge y_i \leq y_o)$.

Proof: The proof is similar to the proof of Lemma 2.

If part: This is the same as the first case of the if part of Lemma 2 proof.

Only if part: Without loss of generality assume that the legal region is not empty but $x_{i_1} < x_{o_1} < x_{i_2}$, i.e. o_1 is on the top side of the die, then $R(\{i_1, i_2\}_{o_1})$ is a point if both i_1 and i_2 are on the top side as well (Figure 8a); it is a line otherwise (Figure 8b). Since $R(\{i_1, i_2\}_{o_1,o_2}) = R(\{i_1, i_2\}_{o_1}) \wedge R(\{i_1, i_2\}_{o_2})$, it is not empty iff $y_{i_1} = y_{i_2}$ and $x_{o_1} = x_{o_2}$, i.e. o_1 and o_2 are at opposite side (Figure 8c). There is no way to add a third output to $\{i_1, i_2\}_{o_1,o_2}$ with a non-empty legal region. Hence, it is a contradiction. ■

By the input-output symmetric nature of legal regions, the above three lemmas apply with the role of m and n interchanged.

Let the condition $(\forall i \forall o\, x_i \geq x_o \wedge y_i \geq y_o) \vee (\forall i \forall o\, x_i \geq x_o \wedge y_i \leq y_o) \vee (\forall i \forall o\, x_i \leq x_o \wedge y_i \geq y_o) \vee (\forall i \forall o\, x_i \leq x_o \wedge y_i \leq y_o)$ be called the *non-overlapping* condition. Then, with these three lemmas, the legality of a node with node placement constraint $\{i_1, i_2, \ldots, i_m\}_{o_1,o_2,\ldots,o_n}$ can be checked with the following algorithm:

1. If n is 1, then the node is legal.
2. If the non-overlapping condition is true, then the node is legal. This can be checked in $O(m+n)$ by first finding the largest and smallest x and y coordinates of both inputs and outputs and then check for the overlapping condition using these values.
3. If the node placement constraint satisfies Condition 2 of Lemma 2, then it is legal.
4. If none of the above are satisfied, then the node is illegal.

It is obvious that this legality checking algorithm is $O(m+n)$. Hence, it is very efficient.

Corollary 5.1 *There exists a corner point* p_c *of* $R(\{i_1, i_2, \ldots, i_m\}_{o_1,o_2,\ldots,o_n})$ *that is closest in distance to all outputs, and a corner point* p_f *furthest from all outputs. The point* p_c *is called the closest point of the region and* p_f *the furthest point.*

Lemma 4 *1. If* $R(\{i_1, i_2, \ldots, i_m\}_{o_1,o_2,\ldots,o_n}) \cap R(\{i_k\}_{o_1,o_2,\ldots,o_n})$, *where* $i_k \notin \{i_1, i_2, \ldots, i_m\}$, *is not empty, then it contains the closest point of* $R(\{i_1, i_2, \ldots, i_m\}_{o_1,o_2,\ldots,o_n})$.

2. If $R(\{i_1, i_2, \ldots, i_m\}_{o_1,o_2,\ldots,o_n}) \cap R(\{i_1, i_2, \ldots, i_m\}_{o_k}$, *where* $o_k \notin \{o_1, o_2, \ldots, o_n\}$, *is not empty, then it contains the furthest point of* $R(\{i_1, i_2, \ldots, i_m\}_{o_1,o_2,\ldots,o_n})$.

Proof: Assume that $m \geq 2$ and $n > 2$. The proof is similar for other cases.

1. Assume $(\forall i \forall o\, x_i > x_o \wedge y_i > y_o)$ (the proofs of the other cases are the same), then $x_k > x_o \wedge y_k > y_o$. If x_k is greater than the x-coordinates of any other input, then $R(\{i_1, i_2, \ldots, i_m\}_{o_1,o_2,\ldots,o_n}) \cap R(\{i_k\}_{o_1,o_2,\ldots,o_n}) = R(\{i_1, i_2, \ldots, i_m\}_{o_1,o_2,\ldots,o_n})$. If x_k is less than the x-coordinate of all other inputs, then the vertical line going through i_k partitions $R(\{i_1, i_2, \ldots, i_m\}_{o_1,o_2,\ldots,o_n})$ into two regions and $R(\{i_1, i_2, \ldots, i_m\}_{o_1,o_2,\ldots,o_n}) \cap R(\{i_k\}_{o_1,o_2,\ldots,o_n})$ is the partition that includes the closest point.
2. The proof is similar to case 1.

■

Lemma 4 says that:

1. Adding inputs to a node placement constraint will not change the closest point of its legal region.
2. Adding outputs to a node placement constraint will not change the farthest point of its legal region.

At this point, we have defined what legal nodes are and how to check for legality of nodes. We now put the legal context into Boolean networks and discuss the implication of legality of Boolean network on placement.

Definition 9 *A Boolean network is* **legal** *is every node in the network is* **legal**.

There is a nice property of a legal Boolean network as described by the following theorem.

Theorem 5.1 *Given a legal boolean network, there exists a monotonic point placement for the network.*

Proof:

This is an induction proof. We traverse the Boolean network in a reverse topological order, i.e. a node is visited only after all its fanouts have been visited.

The base case is where we have all primary outputs. Let o be an arbitrary primary output, then place o at its pin location. For o, its pin location is its closest point. The induction hypothesis is that fanouts of a node n are placed at their closest points and still maintaining monotonicity, i.e. the distances from their closest points to their primary outputs are their Manhattan distances. we show that n can also be placed at its closest point while still maintaining monotonicity.

Let n_f be an arbitrary fanout of n. Let c' be the node placement constraint of n_f with all fanins except n removed. Also let the node placement constraints of n and n_f be c and c_f. Then c_f is derived from c' by adding the primary inputs of fanins of n_f other than n and c is derived from c' by adding the primary outputs of fanouts of n other than n_f. We know that $R(c') \neq 0$ because $c' \subseteq c$ and $R(c) \neq 0$ by the assumption that n is legal. By applying Lemma 4 for each primary input added to c' to form c_f, $R(c_f)$ includes the closest point of $R(c')$. Since $R(c) \subseteq R(c')$, the distance from the closest point of $R(c)$ to a primary output o is the same as the sum of the distance from the closest point of $R(c)$ to the closest point of $R(c_f)$ and the distance from the closest point of $R(c_f)$ and o. Hence, the monotonic property is maintained and n can be placed at the closest point of $R(c)$. ■

Theorem 5.1 reduces our problem of finding a monotonic point placement of a circuit into the problem of finding a legal Boolean network. The logic synthesis transformations we use to convert an illegal Boolean network into a legal one is called *make_legal*, and it is explained below.

5.2.4 Make_Legal

The *make_legal* operation takes a Boolean network as its input and produces a legal Boolean network. In the effort of producing a legal Boolean network, it attempts to minimize the number of new Boolean nodes created.

The following lemma and corollary guarantee that a Boolean network can always be made legal.

Lemma 5 *If $n \rightarrow n_f$, and n is illegal but n_f is legal, then collapsing n into n_f will not make n_f illegal.*

Proof: Collapsing n to n_f does not change the support of n_f, nor does it add any primary output to the transitive fanout of n_f. Therefore, the node placement constraint of n_f does not change and hence n_f stays legal. ■

By the proof of Theorem 5.1, we know that every primary output is legal. Then it is easy to see the following corollary.

Corollary 5.2 *An illegal Boolean network can always be made legal by collapsing all nodes into the primary output nodes.*

Beside collapsing, node duplication can also legalize a node.

Lemma 6 *If $n \rightarrow n_f$, $n \rightarrow n_g$, and n is illegal but both n_f and n_g are legal, then duplicating n into $n_1 \rightarrow n_f$ and $n_2 \rightarrow n_g$ makes n_1 and n_2 legal.*

Proof: The support of n is a subset of both the supports of n_f and n_g, but the output set of the node placement constraint of n is a superset of the node placement constraints of both n_f and n_g. By duplicating n into $n_1 \rightarrow n_f$ and $n_2 \rightarrow n_g$, node placement constraint of n_1 is contained in that of n_f and thus n_1 is legal. Similarly for n_2. ■

Make_legal traverses the Boolean network in a reverse topological order, i.e. a node is visited after all its fanouts have been visited. During the traversal, if it sees an illegal node, it collapses the node into its fanouts until the node becomes legal. Hence, there is a frontier moving from each primary output to primary inputs in its support where every node is legal on the side of the frontier toward the primary output. If the sum-of-product expression of the fanout, as a result of collapsing a node into one of its fanouts, exceeds a user-defined parameter, t, number of literals, the node is replicated for each fanout until it becomes legal. The intuition behind this parameter is that large nodes tend to have more common subfunctions with other nodes and thus allow for sharing. However, the parameter should not be too large since it can result in explosion in memory usage.

As shown above, legality of a node can be checked efficiently, that is, it is linear in the size of the node placement constraint. Hence, the *make_legal* operation is efficient.

5.3 Constraint-Driven Synthesis

The constraint generation step takes a possibly illegal Boolean network and makes it legal. Theorem 5.1 guarantees that there exists a point placement for this network. However, by definition of the point placement of a circuit, nodes are assumed to be a point; hence, they have no area. In reality, nodes have area and the length of a longest path depends strongly on the size of a Boolean network. The constraint-driven synthesis step is responsible for minimizing the area of an already legal Boolean network while preserving its legality. As mentioned in Section 5, we use the number of factored-form literals of a Boolean network as a measure of the area of the circuit represented by the Boolean network. So this step is to optimize the network such that we get a minimum literal legal Boolean network.

We leverage the well developed algebraic transformations in the conventional logic synthesis by extending them to deal with and produce legal Boolean nodes. Each of these operations is explained below.

5.3.1 Fast_Extract

The *fast_extract* algorithm is explained in [16]. It basically looks for a two-cube divisor or a two-literal cube that reduces the most number of literals in every iteration.

When dealing with legal Boolean network, this algorithm may result in illegal divisors. For example, assume that node n is the best divisor found and it divides nodes x, y, and z. Then the output set of the node placement constraint of n is the union of the output sets of the node placement constraints of x, y, and z. From Section 5.2, we know that the legal region of n may be empty and n may therefore be illegal. However, it may be the case that n remains legal if it only divides x and y, or x and z, etc. Hence, the fast_extract algorithm is modified such that the best legal divisor is chosen in every iteration.

If node n divides a set of nodes N, then complexity of finding a subset N_l of N which preserves the legality of n and has the largest reduction in the number of literals is exponential in the size of N. Hence, a heuristic is used to select an optimal subset. First the nodes in N are ordered in decreasing sizes of the legal regions to form a list N_{sorted}. Then N_{sorted} is linearly traversed. Each node is added to the subset N_l if the legality of n is preserved. Node n is used as a divisor if it reduces the number of literals in the network.

In this paper, the fast_extract implemented in SIS is used.

5.3.2 Resubstitution

In the conventional logic synthesis, a node n is resubstituted into another node x if n divides x. This may affect the legality of both n and x. The following observation states when n and x can become illegal.

Observation 1 *If n divides x and both n and x are legal before resubstitution, then after resubstitution*

1. *x can become illegal if its support is not the superset of that of n.*
2. *n can become illegal if its output set is not the superset of that of x.*

In this paper, n can only be resubstituted into x if the legality of n is preserved. Hence, a check is made before every resubstitution.

5.3.3 Full_Simpfily

There are two types of *don't cares*, i.e. the observability don't cares (ODCs) and the satisfiability don't cares (SDCs). Computing the exact ODCs of a node is computationally expensive. In practice, a subset of the ODCs called the compatible ODCs (CODCs) are computed. These CODCs are expressed in terms of the primary inputs. Then together with the external don't cares (XDCs) of the primary outputs, a don't care set in terms of the immediate fanins is computed using an image computation. In computing the SDCs, a support filter is used. A node is included in the SDCs if its support set intersects the support set of the node being considered. Employing SDCs in the minimization procedure can result in boolean resubstitutions. The support filter procedure can also be used in the image computation of the CODCs and XDCs. Once the SDCs are computed and the XDCs and CODCs are expressed in terms of immediate fanins, a two-level minimization algorithm is invoked to find an optimized expression. This is simply a brief description of the *full_simplify*. For a more detail explanation, we refer the readers to [10].

Lemma 7 *Throughout full_simplify computation, the only steps that can introduce illegality into the network are the image computation and the SDC computation.*

Proof: Let node n be the node we are computing don't cares for. Legality of the Boolean network can only change if an edge is added to the network. During the whole full_simplify process, only the fanin edges of n can be added. Edges of fanins of other nodes can not change. Adding a fanin edge to n means that a resubstitution happens and Observation 1 applies. Potential new fanin edges of n are added only during the image computation and SDC computation through the support filter, which basically says that a node x is a potential divisor of n if the support of x intersect the support of n. ■

We therefore constrain this operation by allowing a node x to be in the support filter when computing full_simplify for node n if the inclusion of node x preserves the legality of the network according to Observation 1.

5.3.4 Synthesis Flow

With all the above basic operations, a synthesis flow is then a script similar to the *script.rugged* in SIS. An empirical study needs to be conducted to derive an optimal script.

6 Experimental Results

To see the effect of the proposed approach, we have implemented the basic operations described in Section 5.3. An optimization script has been created and we call it *script.wire*, which consists of:

```
make_legal
eliminate 5
sweep; eliminate -1
simplify -m nocomp
eliminate -1
sweep; eliminate 5
simplify -m nocomp
resub -a
fx
resub -a; sweep
eliminate -1; sweep
full_simplify -m nocomp
```

Our experiment uses SIS and Ritual version 3.4, a timing-driven standard cell placer [12]. The input blif file and a randomly generated pad assignment file is read into SIS. The *script.wire* optimization script is run in SIS to generate an optimized logic netlist. The optimized netlist is mapped to the standard cell technology library *stdcell2_2.genlib* of SIS. The mapped netlist is then placed by Ritual with a fixed pad assignment. We measure the length of the longest path and the delay of the Ritual output. The distance of two cells is measured as the Manhattan distance from the center of both cells. The length of a path is the sum of all distances between consecutive cells along the path.

Table 1 shows the results for four circuits. The circuit *bbaraComb* is obtained from the sequential circuit *bbara* by removing all latches and treating the outputs of the latches as primary inputs and the inputs to the latches as primary outputs of the network. Column 2, 3, and 4 show the number of literals in factored forms of the scripts script.rugged, script.delay, and script.wire respectively. Columns 5, 6, and 7 list the length of the longest path for each script. Columns 8, 9, and 10 show the CPU time. The experiments were run on a DEC AlphaServer 8400 with 2GB of memory. The runtime is for the technology independent step.

As shown in this table, although the number of literals in *script.wire* approach is more than that of *script.rugged*; the length of its longest path is the same for *rd53* and better in other circuits. The longest paths are much shorter than *script.delay* results. As seen from this table, the runtime is comparable. This is expected since the legality checking is linear in the size of the node placement constraints and hence its runtime is a minor part of the total runtime.

Table 2 shows the delay computed by Ritual for the four circuits. Columns 2, 3, and 4 show the cell delay for each script. The wire delay is shown in columns 5, 6, and 7. The total delay is listed in columns 8, 9, and 10. Except for the total delay of *z4ml* running *script.delay*, the total delay of all circuits is the best using *script.wire*.

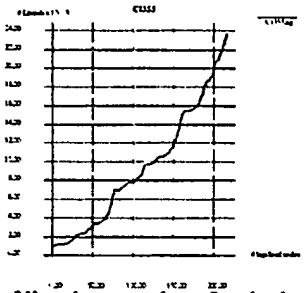

Figure 9: Number of literals vs number of nodes legalized for C1355.

7 Open Issues and Future Work

Though the results in the previous section shows that the approach performs satisfactorily, these circuits are fairly small. For bigger circuits, the number of nodes in a legal network can be large and optimizing such large networks using operations like *fast_extract* and *full_simplify* can be very expensive.

To illustrate this, we plot the number of literals versus the number of nodes in the constraint generation step for C1355 as shown in Figure 9. On the x-axis is the number of illegal nodes that are legalized. On the y-axis is the number of literals in the Boolean network. The network increases from 1032 literals to 23709 literals after 216 nodes have been legalized out of a total of 514 nodes in the network.

There are three various directions that can be pursued to address this problem. The first one is to improve the area optimization algorithm presented in this paper. Rewiring and redundancy removal is a technique that falls into this direction. SPFDs [2] can be used to minimize, rewire circuits, and potentially legalizing nodes. In this paper, we are assuming that we are given a circuit represented as a Boolean network. We then apply make_legal and several algebraic transformations followed by don't care minimization. The final circuit depends on the quality of the initial Boolean network. Alternatively, the Boolean network can first be collapsed as much as possible into a two-level circuit where all primary outputs are expressed in terms primary inputs. Then functional decomposition, like [11], can be used to decompose the network into a minimum literal legal Boolean network.

The second direction which we believe is more promising is to relax the constraint that every path must be monotonic. In other words, this is about solving the slack-based synthesis problem instead of the more restrictive IP-based synthesis problem. This can be done by applying

Table 1: Path length comparison of *script.rugged*, *script.delay*, and *script.wire* for IP-based synthesis.

	Number of Literals			Length of Longest Path			CPU Time		
Name	sc.rugged	sc.delay	sc.wire	sc.rugged	sc.delay	sc.wire	sc.rugged	sc.delay	sc.wire
z4ml	41	84	49	1324	1342	1025	0.2	0.3	0.3
rd53	42	62	50	1122	1624	1122	0.1	0.3	0.2
rd73	74	178	87	1689	2457	1680	0.8	1.8	1.2
bbaraComb	69	79	109	2021	1573	1464	0.5	0.5	0.3

Table 2: Delay comparison of *script.rugged*, *script.delay*, and *script.wire* for IP-based synthesis.

	Cell Delay			Wire Delay			Total Delay		
Name	sc.rugged	sc.delay	sc.wire	sc.rugged	sc.delay	sc.wire	sc.rugged	sc.delay	sc.wire
z4ml	5.66	5.28	4.78	0.93	1.03	0.97	6.59	6.31	5.75
rd53	9.73	7.37	5.94	1.67	2.13	1.42	11.40	9.50	7.36
rd73	7.01	5.09	5.59	1.37	0.88	0.86	8.38	5.97	6.45
bbaraComb	8.18	6.22	4.90	2.19	1.72	1.08	10.37	7.94	5.98

the IP-based synthesis algorithm only to a subset of the paths. Intuitively, we can wireplan only the critical paths so that no diversions are allowed in them; other paths can have diversions. One approach would be to modify the definition of legality so that legality is checked based on the primary inputs and outputs that are relevant only to the critical paths. Only the nodes on the critical paths are legalized. We have done some preliminary experiment and our results show that if you select top few longest paths and legalize all the nodes on those paths, then the area penalty is not very high. However, at present there is no easy way to perform a meaningful comparison of this approach (i.e. modified IP-based algorithm to solve the slack-based synthesis problem) with the conventional approach. For that, we need a placement tool that uses the same delay model as ours and we have not been successful at making Ritual use our model.

One other issue that needs further attention is that of pin assignment The approach in this paper assumes that the pin assignment is given. In the design process, usually only partial pin assignment is given. However, the quality of the final solution strongly depends on the pin locations. Therefore, we need to look into algorithms to find good pin assignment during synthesis. Such an algorithm can also be used to extend this approach to handle sequential circuits by finding good placement for the latches present.

The optimizations that we have shown are technology independent. We have not yet addressed the issue of technology mapping. Also, we have completed ignored gate delays. We are presently looking into both of these issues, i.e. technology mapping and how to best incorporate gate delays in our approach.

Finally we are also looking into extending the proposed approach to handle other interconnect issues, like crosstalk and reliability.

8 Conclusions

We have proposed a new approach to deal with the increasingly importance of wire delays in deep submicron technologies. It is based on the fact that the shortest path between any two points in a circuit is the Manhattan distance between them. We showed an example of why conventional logic synthesis may produce circuits where the minimum distance can not be achieved.

The proposed approach still decouples logic synthesis phase and place & route phase. It consists of a constraint generation step which produces a legal Boolean network, which can be placed such that every path is monotonic, and a constraint-driven synthesis step which minimizes the legal Boolean network while preserving legality. We show an example of how this approach can be extended to solve the slack-based synthesis problem. Finally, we describe directions for future work which includes an investigation into a new placement tool that works together with the proposed approach.

References

[1] Semiconductor Industrial Alliance. *National Technology Roadmap for Semiconductors*. 1997.

[2] R.K. Brayton. Understanding SPFDs: A new method for specifying flexibility. *IWLS*, May 1997.

[3] R.K. Brayton, A.L. Sangiovanni-Vincentelli, and G. Hachtel. Multi-level logic synthesis. *Proceedings of the IEEE*, vol. 78(no. 2):264–300, February 1990.

[4] J. Cong and Z. Pan. Interconnect Performance Estimation Models for Synthesis and Design Planning. In *IWLS 98*.

[5] K. Keutzer, A.R. Newton, and N. Shenoy. The future of logic synthesis and physical design in deep-submicron process geometries. In *ISPD*, pages 218–224, 1997.

[6] Y. Kukimoto and R.K. Brayton. Hierarchical functional timing analysis. In *DAC*, 1998.

[7] R. H. J. M. Otten and R. K. Brayton. Planning for Performance. In *DAC*, June 1998.

[8] M. Pedram and N. Bhat. Layout Driven Logic Restructuring/Decomposition. In *ICCAD*, pages 134–137, November 1991.

[9] M. Pedram and N. Bhat. Layout Driven technology Mapping. In *DAC*, pages 99–105, June 1991.

[10] H. Savoj. *Don't cares in multi-level network optimization*. PhD thesis, University of California, Berkeley, May 1992.

[11] C. Scholl and P. Molitor. Communication based FPGA synthesis for multi-output boolean functions. In *ASP-DAC*, pages 279–288, August 1995.

[12] A. Srinivasan, K. Chaudhary, and E. S. Kuh. Ritual: a performance driven placement algorithm. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 39(11):825–840, November 1992.

[13] G. Stenz, B. M. Riess, B. Rohfleisch, and F. M. Johannes. Timing Driven Placement in Interaction with Netlist Transformations. In *ISPD 97*, Napa Valley, CA, 1997.

[14] H. Vaishnav and M. Pedram. Routability-Driven Fanout Optimization. In *DAC*, pages 230–235, June 1993.

[15] H. Vaishnav and M. Pedram. Minimizing the Routing Cost During Logic Extraction. In *DAC*, pages 70–75, June 1995.

[16] J. Vasudevamurthy and J. Rajski. A method for concurrent decomposition and factorization of Boolean expressions. In *ICCAD*, pages 510–513, November 1990.